Centrum voor Wiskunde en Informatica
REPORT*RAPPORT*

Constraint-based facial animation

Zs. Ruttkay

Information Systems (INS)

INS-R9907 May 31, 1999

Report INS-R9907
ISSN 1386-3681

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Constraint-Based Facial Animation

Zsófia Ruttkay
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

ABSTRACT

Constraints have been traditionally used for computer animation applications to define side conditions for generating synthesized motion according to a standard, usually physically realistic, set of motion equations. The case of facial animation is very different, as no set of motion equations for facial expressions is available. In this paper we discuss a facial animation editor, which uses numerical constraints for two roles: to declare the mimic repertoire of synthetic faces and other requirements a facial animation has to meet, and to aid the animator in the process of composing a specific animation fulfilling the requirements. The editor is thus also a 'motion sculpturing' tool, which lifts the task of creating facial animation from the control data manipulation level to the conceptual design level. The major aid of the editor is to repair inconsistencies due to changes made by the user, and revise changes for which no good repair is possible. Also, reuse of constrained animations, especially expressions, is supported. The main machinery behind these services is interval propagation, which, if using certain type of linear inequalities to express the character- as well as the animation-specific requirements, can produce quickly the interval of feasible values for each control variable. If a solution (usually, repair) has to be produced, it is generated by selecting the best one from a restricted set of acceptable solutions, based on user-defined or automatically generated criteria for the choices.

1991 Computing Reviews Classification System: D.2.2, G.1.3, H.5.1, H.5.2, I.3.6, I.3.8, J.5
Keywords and Phrases: constraint propagation, interval arithmetic, incremental solution update, underconstrained problem, numerical constraints, continuous constraints, animation
Note: This paper has been accepted for publication in the Constraints journal. The work has been carried out under the project INS3.4 'Facial Animation'. An on-line version of this report with some figures in color is available from `ftp://ftp.cwi.nl/pub/CWIreports/INS/INS-R9907.ps.Z`.

1. Introduction

In this paper we show the potentials of an interactive graphical animation editor to produce animations according to requirements of different origin and with different scope, all expressed in the form of constraints. Animation building blocks can be also defined in terms of constraints. The design of such an editor has been motivated by the objectives of the 'Facial Analysis and Synthesis of Expressions' project (FASE 1998) to produce realistic 3D (Parke et al. 1996) and cartoon-like 2D facial animations (Brennan 1985), therefore we will use facial animation as the working example. The issues of facial animation differ in many respects from those of animating the body or controlling and generating motion for kinematic systems. However, the ideas behind the facial animation editor are general and can be applied to other animation tasks as well, where change of form/position in 2D or 3D has to be controlled directly, via parameters.

In the rest of the introduction we discuss the specific issues of facial animation and explain motion sculpturing as the basic idea behind our facial animation editor. We also compare our approach to other paradigms of animation editing, to graphical user interfaces and to musical composition applications. In Section 2 we introduce how animations are represented, and how expressions and building blocks are defined in terms of constraints. We discuss how other requirements, with different source and scope can be expressed by sets of constraints. In Section 3 the issues of constraint processing are dealt with. First of all, we pinpoint the expectations originating from the role of constraints in

animation editing tasks. Then we introduce the interval constraint propagation mechanism and show how it is used as a basic engine to reduce the domains, to generate instances of building blocks and to repair non-solutions. Finally the current system and features under implementation are described. The paper is concluded by outlining issues of further development and by summing up novel features of our system.

1.1 Facial animation

Computer facial animation has been a flourishing research topic for more than 25 years, aiming at building models of human faces which can be animated and used to (re-)produce facial expressions reflecting emotions and mouth movements for spoken text. The bulk of the efforts has been spent on producing models which can be deformed to shapes characteristic of human facial expressions. The majority of the research is concentrated on (re-)producing realistic human faces, urged by the needs of such applications as televideo, teleconferencing, facial surgery and naturally those of the film and entertainment industry such as lip synchronization and synthesized actors. Next to these, there is another set of applications where non-realistic human-like faces have to be animated, not faithfully but in an expressive and appealing way. Applications of this type include animation films and the so-called 'social user interfaces': human-like creatures guiding the user in using a software or in walking through a virtual reality environment, or reflecting the state of a system via facial expressions. For the first type of applications sophisticated 3D models are required, while for many of the second type applications simple, often 2D models are also sufficient. The mechanisms used to deform the 3D and 2D models may be very different, but the issues of how to define a proper animation are similar and not yet properly solved for either of the cases. To make our point clear and provide background for the animation editor, we shortly outline two representatives of the 3D and 2D models. These are also the ones we have implemented and have animated with the help of the editor to be discussed.

The so-called *physically-based models* (Terzopoulos et al. 1993) use a multi-layered elastic mesh with simulated muscles. The deformation of the face is given in terms of contraction of the individual muscles. Depending on elasticity and width/length parameters of the muscles, forces arise directly due to the muscle contraction on some of the nodes, and these forces are propagated along the elastic mesh. According to the laws of dynamics, the nodes of the mesh move to a new equilibrium position. With additional constraints, for instance that nodes corresponding to points in the layers of the facial tissue cannot penetrate the underlying skull and organs, or that the total volume of the facial tissue is preserved, the mechanism of the muscle-driven deformation of a synthetic face comes close to the physical reality. Such 'minor' problems as the proper parameterization of the individual muscles, the elasticity characteristics of the tissue layers, conformation of a 'generic face' to an individual one (not forgetting about such aspects as varying tissue width) are still to be solved to be able to come up with a faithfully synthesized model of a given human face. In Figure 1. our 3D **Persona model** is illustrated, for technical description see (Persona 1998). Note that physically-based models can be built for non-human and even non-realistic heads, such as animals or a tea-pot. A common approach is to control the deformation of these faces by specifying the contraction of the muscles built in the model.

In case of 2D *cartoon-like models* (Litwinowicz 1991, Van Reeth 1996, Thórisson 1996) features of the face can change shape and/or position. We have developed the **CharToon Face Editor** to define faces with deformable features. Features can be moved and deformed directly or indirectly, via a hidden skeleton-like inner structure, by specifying the value of several controller points, each of which can move in the plain within a predefined rectangle. The deformation of such a face is defined by the position of all the controller points. In Figure 2. an example is shown, more details on CharToon can be found at (CharToon 1996).

1.2 Sculpturing facial animations

Parallel to the emergence of improved realistic models, one has been confronted with the fact that there is neither enough knowledge on the dynamism of human facial expressions, nor appropriate

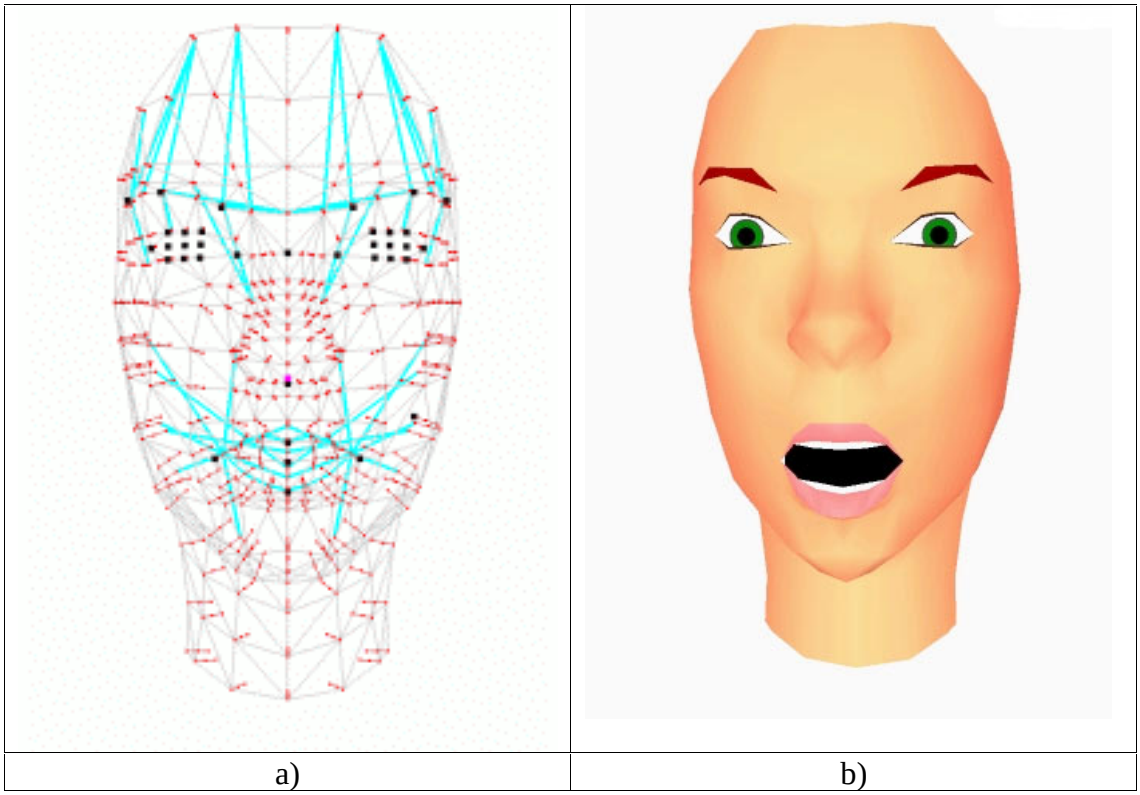


Figure 1: a) A physically-based Persona face: the nodes and connecting springs of the 3 layers and the muscles. b) The rendered surface.

Figure 2: A 2D CharToon face with the controllers used for deformations of features. One or two-dimensional cross-hairs indicate range of the location of components or of points defining the shape of a component. Some examples: Tears can move vertically, pupils can move within a rectangle. The shape of the mouth is defined by a skeleton with two-dimensional controllers at the lip corners and by one-dimensional ones at the middle of the upper and of the lower lip.

paradigms and tools to animate synthetic faces.

Make a smile!

To illustrate the problem, let's assume that there is a perfect physically-based facial model at our disposal with a sufficient number (like 10-15 pairs) of facial muscles corresponding to the subset of the real muscles most involved in facial expressions. Our task is to make this face smile. In order to simplify our discussion, we restrict the task to defining the contraction of the most important muscle pair involved, the Zygomatic major muscles, pulling up diagonally the corners of the mouth. We want to produce not only a 'human smile', but the smile typical of the person — real or invented — in question. We will use a further simplifying hypothesis, namely that the muscle activation happens in three, linear stages: *application, release* and *relaxation* as given in [htbp] 3. (It has been shown experimentally that the actual shape is far more complex (Essa 1994), but because of the lack of sufficient evidence on the real shape, trapezoid-shaped functions have been widely used.)

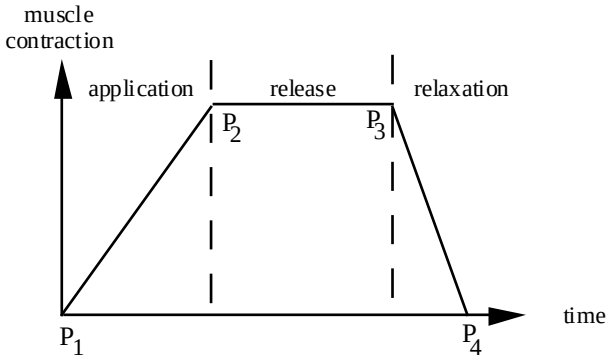


Figure 3: Stages of contraction of a muscle (based on simplifying assumption).

One can define infinitely many pairs of trapezoid-shaped muscle contraction functions. But which ones produce an acceptable smile? How short or long should a smile last? How are the duration of application, release and relaxation related? (It has been observed that in case of real smiles of different length, the three time intervals do not scale uniformly.) 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 expressions may and may not occur while smiling? What is the total effect of co-existing expressions (e.g. smile and speech)?

Motivations for animation sculpturing

One may conclude that the questions above are to be answered by analyzing a huge sample of real smiles, and that with the development of face motion capturing hardware and software a faithful animation could be and should be done on a performer-driven basis (Essa et al. 1996, Williams 1990).

There is research going on to accomplish the first task (Ekman et al. 1978, Essa 1994, Guenter et al. 1998). However, the problem seems to be very hard to tackle: it is difficult to get enough real, spontaneous facial expression samples recorded under circumstances needed for analysis, and the computation of the contraction value of the individual muscles based on observed facial deformations is not well established. Hence, an environment to allow guided experiments with invented synthesized expressions may help the process of learning about the laws of real expressions.

On the other hand, the declarative definition of the facial repertoire of a character would be very helpful when inventing animations for realistic or cartoon-like faces. One would like to be able to declare such characteristics of the facial repertoire as permanent or conditional (e.g. in case of excitement) asymmetries in the motion of pairs of features, or the typical way of smiling. These characteristics

would be automatically enforced, and predefined building blocks could be reused in course of editing an animation for the face. Also, when editing a particular animation, the user would have tools to add and modify requirements. Thus such a tool would make the presently very labor-intensive and low-level process of creating animations easier and faster, and lift it to a higher conceptual level. The reuse of pieces of animation could be also supported, by adjusting an animation to meet a modified set of requirements. Thus the facial animation editing tool we envision has two intended usages:

- **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.

Characteristics of the animation application domain

The questions the animator has to answer when producing expressions are essential and difficult whatever facial model one has to animate, and the related issues are basically independent of the model in question. One can define several different mimic repertoires for the same facial model, as the facial model itself does allow a huge variety of deformations and any sequence of them.

This is a big difference compared to other domains of animations, where generic physical laws of motion and given physical properties of the model — e.g. size, weight, maximum angle of joints — are used to compute motion characteristics of the body, based on some given control values, such as motion parameters of some parts of the model. In case of facial models the motion of feature-points of the face are basically independent. For cartoon-like faces, this is an advantage: the animator would like to experiment with unrealistic facial expressions (like eyebrows jumping off the head, eyes growing big). In case of realistic facial models, this is partly a deficiency of the model, due to the fact that little is known about the anatomy-based co-articulation and physiology of the muscles. As far as we know, no physically-based facial model has been made to reproduce the phenomenon of muscle co-articulation based on the physics of the model. Note, however, that much of the muscle co-articulation is not caused by the anatomical structure of the face.

On the other hand, in case of physically based models there is a different task, which more or less corresponds to the usual inverse dynamical motion control problem. If one specifies the 3D location and possibly other motion parameters of certain feature points on the face (corresponding to certain vertices on the upper layer of the facial polygon), then the motion parameters of the rest of the vertices of the multi-layered mesh of springs and the contraction of the muscles have to be updated, based on the dynamical laws of motion. Note that the aim here is to reproduce single deformations, not to synthesize facial motion.

In the better explored field of body animation, the concepts corresponding to that of facial expressions like ‘a big smile’ or typical mimic characteristics as ‘less articulated right-eyebrow movement’ would correspond to ‘a happy jump’ (may be not high, but with hands above the head, though one could jump equally high with hanging hands) and to ‘limping walk’ (right steps are always smaller than left steps, though the body is symmetrical). These are motion characteristics which cannot be derived from the physics of the body. In general, the functionalities of our facial animation editor could be used in animation domains where a big number of the control parameters are independent of each other, both concerning the state at a given snapshot as well as in time. In such domains it remains for the animator to sculpt the dynamism of the model to be animated. Cartoon-like (Owen et al. 1994) and emotional body animation are such fields.

1.3 Comparison to other approaches

Parameter keyframing has been a common practice because of two reasons: it provides complete freedom for the animator, and also, often because of the lack of more powerful tools for many animation tasks. Our editor can be seen as extension to parameter keyframing. The first difference is that not all parameters have to be specified for each keyframe. The second and more important extension is that constraints provide a basis for declarative keyframing and the definition of building blocks.

Most of the commercial animation packages do allow the manipulation of **motion curves**. Manipulation is usually restricted to one channel at a time (corresponding to location and speed co-ordinates). In the latest version of Alias Wavefront (Alias 1998), it is possible to define functional constraints between channels. However, constraints are treated in an ad hoc way and have a role only in generating animation data. The idea of using constraints to characterize the required motion or to define building blocks is missing from Wavefront.

The recent FaceWorks software (FaceWorks 1998), particularly designed as an authoring tool for facial animation, is in many respects limiting. Only expressions can be manipulated, namely inserted/deleted/scaled, animations cannot be fine-tuned on parameter level. The intensity and duration of expressions can be changed. Though the definitions of the provided expressions are hidden, it looks like that there are no constraints used as in our case. E.g. expressions are scaled linearly, and thus arbitrary short/long expression actions can be generated. Neither variants of an expression can be generated, nor new, person-specific expressions can be defined and used as building blocks. The user has no means to define requirements, in our terminology only animation data can be specified.

Software packages for **performer-driven animation** address the issue of modifying and reusing performer data. The recent Famous software (Famous 1998)is similar to our editor in allowing transformations or fine-level editing of selected tracks and time intervals of performer data. The system ensures that a set keyframe is 'smoothly interpolated' to recorded data preceding/following the keyframe. However, the main difference is that there are no constraints used, the operations are performed as Bezier curve operations. Closest to our approach is the body motion capture data system outlined in (Da Silva et al. 1997), which provides a framework and a good interactive environment to process and reuse curves obtained by motion capture. Concatenation and blending of motions is based on mechanism for processing signals. In both systems, building blocks are pieces of animation data, in our terminology, not constrained animations. All the same, the success of these techniques indicate the urging need for tools to manipulate and reuse pieces of animations.

Motion warping techniques (Witkin et al. 1995) and signal processing based motion curve transformations (Bruderlin et al. 1995) have similarities to our approach, namely that they transform motion in a controlled way. The principles behind these techniques are often intuitive and can be given in qualitative terms rather than in terms of strictly defined characteristics, and always concrete curves are manipulated. (No motion types can be declared and instantiated.) They do not allow as fine control as one has with constraints. An exception is the work on constraint-based motion adaptation (Gleicher et al. 1996), which uses the combination of motion signal processing methods and constraint-based direct manipulation, in order to be able to modify an existing motion to meet certain requirements. The requirements are expressed in terms of certain types of constraints on new values for some parameter channels at arbitrary time moments, not only at control vertices defining the time curve, like in our case. The 'best' perturbed motion is the one where the total change of parameter values of control vertices is minimal. Hence, in contrast to our approach, the time of control vertices cannot be changed. The strength of the system consists in that the locally prescribed constraints have an effect in some sense on the entire motion on the basis of trying to keep the perturbed motion similar to the original one. Our approach differs in using explicit constraint propagation to propagate effects of local modifications, and allowing the user to control dynamically the range of propagation. It would be interesting to see how similar perturbed motions are, if generated by the two methods. Our expectation is that if applied for sequences of facial expressions with fixed times and loose expression definition constraints, the perturbed motions will be close to each other. The concept of building blocks defined by constraints as well as more sophisticated requirements than one-time constraints are missing from Gleicher's and Litwinowicz's work. They remain on the data level use of one-time constraints, in accordance with their primary goal of being able to perturb animation data. Their approach, just like ours, is not restricted to modifying physically correct motions.

There is extensive literature on **motion synthesis** and **motion control** systems based on some general constraints and principles of physical motion. Many systems apply dynamical constraints (Hodgins et al. 1995, Kokkevis et al. 1996, Witkin et al. 1990), which are universal constraints ex-

pressing the Newton laws for the motion and deformation of real objects. In the case of inverse kinematics general motion and geometry equations and constraints can be used. In both cases, the 'environment' can be modified by changing relevant parameters of the model (e.g. mass and geometry of the person walking, limits on relative extreme positions of moving parts) or by prescribing the value of some of the motion parameters (location, velocity) at certain times (Cohen 1992). In these cases, constraints are used to generate a piece of motion, which is a more limited usage than in our case.

Finally, the user interface of our animation editor brings into mind graphical user interfaces (Borning et al. 1995, 1996, 1998), layout systems (Oster et al. 1998, Vander Zanden et al. 1995, Pachet et al. 1998) and systems supporting musical composition (Pachet 1999), which apply constraints and often use some specific incremental propagation method to update solutions. What makes our animation editor basically different from **layout systems** is that the parameter staves are only a visual representation of an animation. Also, the animation and requirements are defined in terms of parameters possibly without any geometrical meaning. All the same, the particular visual representation is chosen because it is suitable to show important characteristics of the animation. Hence the editor can be considered as a **graphical user interface** with unusual specific features (e.g. role of time line, possibility to change constraints in several disjunct areas at a time).

It is an interesting question if the somewhat similar notation and the critical role of time in **musical composition** systems could provide applicable techniques for animation editing. There are similarities in using constraints in a declarative way, such as the choice of definitions of locality for perturbations and the need for criteria for selecting from a huge set of solutions. However, in the musical composition domain, there exists a canonized knowledge on musical styles, which is supposed to be part of the knowledge of the composer and/or of the musical composition system. E.g. if a 'Wiener minuet' is to be composed, there are criteria on the meter, the structure and the harmony to be met. Moreover, a single well-established musical notation can be used to compose all kinds of music. In the case of facial animation there is much freedom in defining general requirements and building blocks, there are neither accepted 'styles' nor a language to define them. The lack of domain-specific knowledge and notational conventions make the task of designing an editor for facial animation especially challenging.

2. Constrained animations

In this section we explain how constraints can be used to define reusable facial expressions as well as different requirements concerning the animation. In the rest of the paper we will use the earlier introduced simplified smile as an example. One should remember that usually the animator has to orchestrate dozens of parameters, and define the value of each parameter for each 40 millisecond (assuming 25 frames per second frame rate for the animation).

2.1 Types and representation of animation constraints

A *deformation* of a face is defined in terms of a fixed number N of parameter values. Each *parameter* may take its value from a domain of closed 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 temporal evolution of the deformation along time. An animation is given by the vector of functions $(F_1(t), \dots F_N(t))$, where $F_i : T \to D_i$ 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, similarly to the idea of traditional parameter inbetweening. (Inbetweening is the common practice of making animations by defining the position/shape of a character for some, so-called keyframes only, and the position/shape for frames between keyframes is derived on the basis of the keyframes before and after the frame in question. In the simplest case linear interpolation is used, but human animators — and some of the computer animation software packages too — have a broad repertoire of other principles.) In our discussion, we will assume that the not explicitly given parameter values are computed by applying piece-wise *linear interpolation* on the intervals between the time moments with given parameter values. Our

approach is insensitive to what particular interpolation method is used. As there is not yet sufficient data about the characteristics of facial parameter curves, we have no reason to use some specific type of interpolation. However, one could use e.g. cubic polynomials to get smooth interpolating curves. We will use the notion *parameter* curve for the graph of a parameter function.

As introduced above, for the i-th parameter channel, a number of $P_j^i = (t_j^i, v_j^i)$ *control points (CPs)* are given which define the parameter curve for the channel. The number of control points may differ from channel to channel, and control points for different channels need not be aligned along time. We assume that control points of each channel are indexed according to increasing time, that is $t_j^i < t_{j+1}^i$. If for a channel no control points are given, then the value of the parameter is assumed to be the neutral value. We will refer to the co-ordinates of a CP as the *time* and *parameter variables*, and to their values as the time and the parameter value of the CP. We will also talk about the CP at a given time, the CPs within a time interval, and the preceding and following neighbor of a given CP.

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 $(F_1(t), \dots F_N(t))$ functions together produce an animation with the requested characteristics. We deal with requirements which can be expressed in terms of certain types of *constraints on co-ordinates of control points*. All the allowed types of constraints limit the value of a function of co-ordinates of certain control points. We will use extended intervals to indicate these limits: $I = [\underline{I}, \overline{I}]$ is a finite or infinite interval, the $\underline{I}$ and $\overline{I}$ are reals or $\pm\infty$, and $\underline{I} \leq \overline{I}$. $\mathcal{I}$ denotes the set of all extended intervals, while $\mathcal{I}'$ denotes the intervals of $\mathcal{I}$ not containing 0 in their inside, and $\mathcal{I}^*$ denotes the intervals of $\mathcal{I}'$ not containing $\pm\infty$. Defining the $\leq$ relation for the extended reals, this notation allows inequalities and equalities to be expressed in the form of membership in extended intervals. E.g. $x - y \geq 20$ will be expressed as $x - y \in [20, +\infty]$.

In general, all constraints of the form $c(x_1, ...x_p) \in I$ are allowed for which the $c : R^p \to R$ *constraint function* is continuous and monotone in each variable on the domains of the variables. For the rest of the paper we assume that only so-called *basic facial animation constraints* are used. The basic facial animation constraints are all linear. The unary constraints limit the domains for variables, the binary ones limit the difference of time or parameter variables or the proportion of parameter variables. The 4-ary constraints limit the proportion of time intervals between two pairs of CPs of the same channel, or the proportion of value and time difference between two consecutive CPs. These constraints are expressive enough to formulate a range of animation requirements concerning synchronization, intensity and duration of expressions, and speed of appearance. The basic facial animation constraint types are listed in Table 1.

Table 1: The types of basic facial animation constraints.

(Ia)	$t_j^i \in I$	time range	(Ib)	$v_j^i \in I$	value range
(IIa)	$t_j^i - t_m^n \in I$	time duration	(IIb)	$v_j^i - v_m^n \in I$	value range
(III)	$\frac{t_j^i - t_k^i}{t_r^i - t_s^i} \in I$	relative time duration, where $I \in \mathcal{I}'$			
(IV)	$\frac{v_j^i}{v_m^n} \in I$	relative parameter value, where $I \in \mathcal{I}'$			
(V)	$\frac{v_{j+1}^i - v_j^i}{t_{j+1}^i - t_j^i} \in I$	parameter change speed			

An animation A is given as a sequence of control points for each parameter channel. This data is sufficient to play the animation. However, if the animation is to be altered, then one needs to know about the constraints which express the requirements the animation is supposed to meet. An animation with a set of constraints is called a *constrained animation,* and is denoted by the tuple $< A, C >$. The variables of C are the co-ordinates of the CPs in A. An animation without constraints will be also called as *animation data.* We will denote the CPs referred to by C as $cp(C)$, and the variables as $vars(C)$. A constrained animation is *partial,* if not all the variables (co-ordinates of CPs) have an assigned value, otherwise it is *complete.* A constrained animation is *feasible,* if all the constraints in C, which refer to only instantiated variables, are satisfied. A *good animation* is a complete and feasible constrained animation, that is one where A is a solution of C.

2.2 Definition of expressions

It is common practice of animators to reuse some earlier made pieces — such as a smile, a blink and a mouth-shape — as building blocks. By defining building blocks as constrained animations, it is not only possible to generate and paste proper animation data, but additionally to manipulate the pasted data in accordance with the constraints given for the building blocks. We will refer to constrained animations to be used as building blocks as *expressions,* and to animation data that are a solution of the constraints prescribed for the expression as *expression actions.* (Expressions are used in the broadest sense, not only for animations with the semantics of real facial expressions.) A particular solution represents the so-called *default expression.*

We keep the notion *snapshot* for certain static configurations, that is expressions with at most one CP for each parameter channel, and all CPs with the same time value. Similarly to expressions, snapshots are defined by constraints on (some of the) parameter variables. Hence, the smile expression defines the dynamic process of smiling, while the smile snapshot defines a frozen smile.

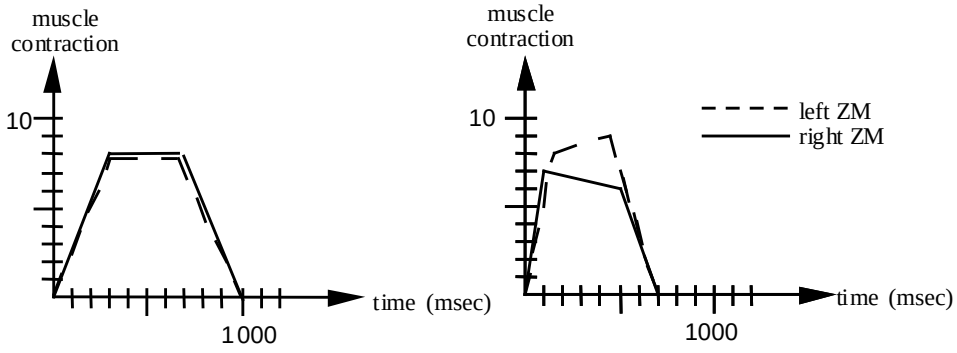


Figure 4: Two smile actions which are solutions for the same set of constraints.

EXAMPLE 1 *A smile expression and two smile actions.*
In Table2 the definition of a smile expression is given. Only unary constraints and binary constraints of type II are used. The binary constraints (1a,b), (2a,b)and (3a,b) provide limits for the duration of the application, release and relaxation stages, (4a,b) and (5a,b) tell how the timing of the activation of the two muscles should be synchronized. Particularly (4a,b) declare that the activation of the two muscles should start and end at the same time, while for the other two control points some deviation is allowed. The constraints (9a,b) limit the difference between the values of the corresponding control points for the two muscles. Finally, constraint (10) tells how much the values may differ at the beginning and at the end of the relaxation of one of the muscles. The given set of constraints has many solutions, each corresponding to a particular smile action. In Figure 4 the parameter curves for two smile actions are shown.

Table 2: Constraints of the smile action. For $j = 1, \ldots 4$ (t_j^1, v_j^1) are the control points defining the contraction function for the right, (t_j^2, v_j^2) for the left Zygomatic major muscle.

(0a)	$t_j^i \in [0, 30000]$	(0Ib)	$v_j^i \in [0, 10]$
(1a)	$t_2^1 - t_1^1 \in [50, 300]$	(1b)	$t_2^2 - t_1^2 \in [50, 300]$
(2a)	$t_3^1 - t_2^1 \in [100, 400]$	(2b)	$t_3^2 - t_2^2 \in [100, 400]$
(3a)	$t_4^1 - t_3^1 \in [100, 300]$	(3b)	$t_4^2 - t_3^2 \in [100, 400]$
(4a)	$t_1^1 - t_1^2 \in [0, 0]$	(4b)	$t_4^1 - t_4^2 \in [0, 0]$
(5a)	$t_2^1 - t_2^2 \in [-50, 50]$	(5b)	$t_3^1 - t_3^2 \in [-50, 50]$
(6a)	$v_1^1 \in [0, 0]$	(6b)	$v_1^2 \in [0, 0]$
(7a)	$v_4^1 \in [0, 0]$	(7b)	$v_4^2 \in [0, 0]$
(8a)	$v_2^1 \in [7, 10]$	(8b)	$v_2^2 \in [7, 10]$
(9a)	$v_3^1 - v_2^1 \in [-1, 1]$	(9b)	$v_3^2 - v_2^2 \in [-1, 1]$
(10)	$v_3^1 - v_4^1 \in [5, 8]$		

2.3 Expressing requirements by constraints

The definition of expressions is only one possibility for the declarative usage of constraints. Below we look at how other requirements can be expressed in terms of constraints, and how the constraints for an animation can be generated, based on the scope and origin of the requirements. Without going into details, posting a requirement on a constrained animation $< A, C >$ results in a new, maybe partially constrained animation $< A', C' >$. Note that nothing is said about how new CPs are added, and if the new animation A' is feasible. Often $cp(C') = cp(C)$ and $C' \supset C$, that is, by adding a requirement only constraints are added.

EXAMPLE 2 *Symmetrical facial motion.*
Often one wants to generate symmetrical motion of the face. This requirement has two effects on the CPs of the corresponding parameter channels of the left- and right features:

- *the number of CPs should be the same for the two parameter channels;*
- *the 1st, 2nd, ... CPs should have the same time- and value for both channels.*

Let us examine the effect of this requirement on a constrained animation $< A, C >$, particularly on the 1st and 2nd channels corresponding to feature pairs. Possibly new CPs are added to the ones in A to make the number of CPs equal for the two channels, and the $t_j^1 - t_j^2 \in [0, 0]$ and $v_j^1 - v_j^2 \in [0, 0]$ constraints are added to C, for $j = 1, 2, \ldots M$, where M is the number of CPs in the channel 1 (and also for channel 2).

Scope of requirements

Requirements may be posed for different time intervals and/or channels. This aspect is expressed by the *scope* of the corresponding constraint, which can be one of following:

- **General:** The constraint should hold throughout the entire animation and for all parameter channels.
- **A single parameter channel:** The constraint should hold for the entire duration of the animation for one parameter channel.
- **One expression:** The constraints should hold for control points of all expression actions of a certain kind.
- **Certain parameter channels:** Two or more parameter channels are coupled, in a given time range or for the entire duration of the animation.
- **Local:** One-time specific constraint for a selected set of control points.

Source of requirements

When working on an animation, the requirements and resulting constraints to be taken into account are from different *sources*:

- **The 'physical limitations'** of the face to be animated, such as muscle contraction (speed and value) limits. Note that the 'physical limitations' may reflect the (assumed) anatomical characteristics of the face, but could be of other nature, such as limitations of the rendering to be used.
- **The behavioral repertoire** of the character to be animated, such as articulated eyebrow movement, as typical for the character.
- **The storyboard** of the animation, such as requirement for lip-synch according to written text or recorded speech to be spoken by the character.
- **The animator** who may add further, global or local constraints e.g. for synchronization, or for refining an expression action.

On the basis of the scope and source of the requirements it is possible to pose/withdraw several requirements together, and thus add/remove sets of constraints. E.g. as soon as a particular face is to be animated, the requirements associated with the 'physical limitations' of the face are posed. If audio of spoken text is also given, then the requirements on mouth shapes at certain time moments are posed. The two sets of requirements can be independently withdrawn and posed. We do not discuss the technical details of requirement posting in this paper.

3. Constraint processing while editing

Editing an animation takes place by a sequence of two kinds of editing operations, which can be performed by directly manipulating a graphical representation of the control points of the parameter functions:

- **adding/deleting** (groups of) CPs;
- **changing parameter and/or time value** of (groups of) CPs.

Interwoven with the manipulation of control points, the animator also changes the set of constraints in two ways:

- **implicitly,** as the addition/deletion of a CP usually implies the addition/retraction of constraints (and eventually also, the addition/deletion of further CPs), due to the scope and source of requirements (see Figure 5.);
- **explicitly,** constraints may be added/deleted/modified in a direct way, by changing requirements of all possible sources.

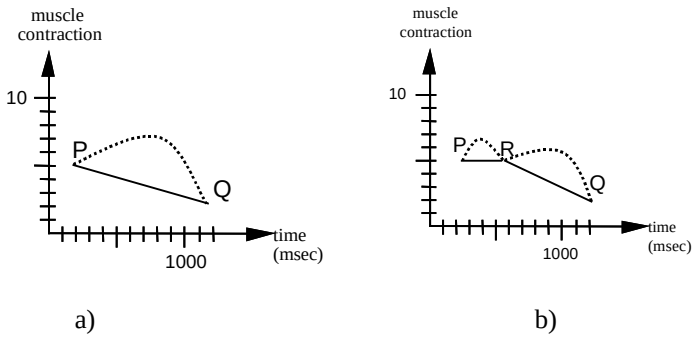


Figure 5: The arcs indicate binary constraints, expressing temporal ordering between neighboring CPs. In a), there is a binary constraint between CPs P and Q. In b) a third CP, R has been inserted between P and Q. The insertion of R implied removal of the initial binary constraint between P and Q and the addition of two new ones.

A constrained animation should be good, that is it should be a solution of the current set of constraints. When the user manipulates the current good animation, goodness may not hold any more. As a response to the user's manipulation of the animation A or the set of requirements R (and thus, set of constraints C), the animation is perturbed to one which is good, that is a solution of the updated set of constraints. Perturbation is understood as generating a solution that is acceptably close to the original animation. If there are no acceptable solutions, the latest good state is restored. Otherwise, a best acceptable solution — closest to the original animation — is selected. The scenarios are given below:

```
1. procedure Change_Animation(A, C, R, A*)
2. A'← Add_Delete_CPs(A*, R)
3. C'← Add_Delete_Constraints(A', C, R*)
4. if Acceptable_Solutions(A', C', A) = ∅
5.       then return(A, C, R)
6.       else return (Best_Solution(A', C', A), C', R)
7. end
```

```
1. procedure Change_Requirement(A, C, R, R*)
2. A'← Add_Delete_CPs(A, R*)
3. C'← Add_Delete_Constraints(A', C, R*)
4. if Acceptable_Solutions(A', C', A) = ∅
5.       then return(A, C, R)
6.       else return(Best_Solution(A', C', A), C', R*)
7. end
```

3.1 Constraint handling expectations

The challenge is to provide an animation editor which works according to the above outlined scenarios. The constraint handling mechanism of the editor has to meet the following expectations:

- **Fast response:** the solution method should be fast enough to provide real-time repair of the current animation as a response for user input.
- **Feedback on feasible regions:** the feasible region should be made visible for the user to guide him while manipulating the animation.
- **Dynamic restrictions of the solution space:** different criteria to restrict the search should be dynamically generated and taken into account.
- **Preferences for solutions:** if many solutions exist, the best one should be selected, with respect to automatically generated or user-defined preferences.
- **Incremental update:** as a response to frequent but small modifications of the current good animation data and of the constraints by the user, the animation data has to be repaired/extended, by reusing much of the latest good animation.
- **Help in case of conflicts:** some mechanisms should be available to point out conflicts and to guide the user in resolving them.

The CSPs used to define animations exhibit many characteristics as listed below, which restrict the choice of applicable solution methods. All but the last two are common in interactive graphical systems (Borning et al. 1998).

- **Non-functional constraints:** if all but one variables of a constraint are instantiated, there can be more than a single possible value chosen for the remaining variable in such a way that the constraint is satisfied.
- **Multi-way constraints:** there is no general input-output role assigned to the variables of a constraint. A cast of role can be assigned dynamically, depending on the semantics of the variables and the specific interaction of the user.
- **Cycles may occur:** there may be cycles in the constraint graph. However, cycles are usually short and the constraint graph is not dense.
- **Numerical infinite domains:** the domains are infinite, specifically intervals of reals or integers.
- **Variables change dynamically:** the set of variables is not known in advance, but changes due to editing.
- **Ordering by time:** the time of CPs provides a basis for defining distances of CPs, which can be used to focus on sub-parts of the entire problem and to define ordering for variable instantiations.

Finally, the type of constraints and the requirements to be used for different animations cannot be given in advance, which poses further demands:

- **Robustness with respect to constraint types:** the above listed characteristics of the constraint handling should be valid for a wide range of constraint types, not only for the basic ones.
- **Tools to edit requirements and constraints:** Constraints and requirements should be defined and manipulated on different levels, e.g. switching on a requirement for a selected time interval, defining an individual constraint on some of the CPs. Particularly, visualization of constraints and mechanism to name variables are difficult UI issues.

3.2 Reduction of interval domains

Interval propagation is a powerful and general paradigm for reducing numerical interval domains (Benhamou et al. 1997, Lhomme 1993), and can be well adapted to fulfil most of the requirements we listed. In a nutshell, interval propagation algorithms iteratively compute tighter and tighter bounding boxes — direct product of intervals — around each solution, based on the idea of splitting a selected domain and tightening the domains of the rest of the variables for each split half. The idea of propagating values of the bounds is close to partial lookahead for finite domains as defined by (Van Hentenryck 1989). There exist current efficient systems that use the idea of interval propagation for computing solutions (Benhamou et al. 1994, Van Hentenryck 1997, Van Hentenryck et al. 1998). However, the following points have to be taken into account before opting for such an algorithm:

- **No solution is generated:** After each step, approximating boxes around the solutions are provided. In general, the user knows nothing about the number and distribution of all the solutions within the bounding box. Hence the task of generating a single exact solution is beyond the capability of these algorithms. Moreover, if there are many solutions far from each other, then the big number of small-size bounding boxes may cause a combinatorial explosion.
- **Propagation of intervals may be costly:** intervals are tightened step by step, on the basis of estimating the value of a constraint on boxes. In case of complex, non-linear constraints this may require some expensive computation, making the algorithm slow.

In facial animation editing the above shortcomings of interval propagation can be avoided by such compromises which do not limit the expected functionalities. Namely, our approach is based on two assumptions:

- The projection of any constraint on each variable, restricted to the intersection of any box and of the solution set, is a closed interval.
- The projection of each constraint on each variable, restricted to the intersection of any box and of the solution set, can be computed fast.

If all the constraint functions are continuous and monotone in each variable, both criteria are met. We believe that such types of constraints are sufficient to express all kinds of requirements arising in facial animation.

From the point of view of direct manipulation the assumption of intervals without holes as the projections of the solution set is reasonable. Intervals can be easily shown to the user to guide him to remain within the feasible region when dragging a CP, while it would be hard to make a 'jumping over holes' service transparent.

Interval propagation can be done for uninstantiated variables of partial solutions, by propagating the known value of the instantiated variables. The decision of how to instantiate free variables — in which order, and to what particular value within the allowed interval — is left for the user, or for some automatic (but tunable) mechanism to guide the search. This is actually very much suited for interactive editing, where the user initiates changes of variable values. There are usually many solutions, so to be able to define flexible strategies to find preferred solutions is an advantage, not a burden.

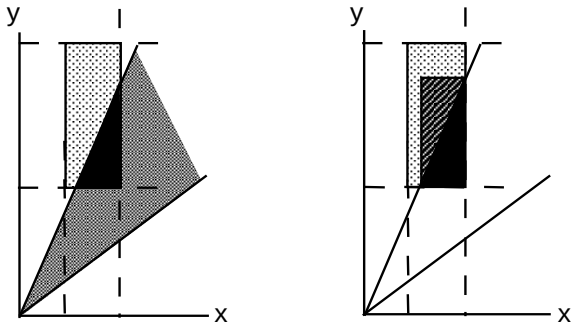
Domain reduction can be also performed on the domain of possible values for the constraint functions. Namely, finding the solutions of a p-ary constraint of the type $\underline{I} \leq c(x_1, \ldots x_p) \leq \overline{I}$ is equivalent to finding the roots of the function $f_c(x_1, \ldots x_p, y) = c(x_1, \ldots x_p) - y$, where the allowed domain for y is $[\underline{I}, \overline{I}]$. The reduction of the domains of y, that is of the value of the constraint functions, provides valuable information for coping with changes of constraints, to be discussed in 3.5.

Interval propagation reduces the domains on the basis of *interval reduction functions* associated with constraints, and propagates the changes. The process is repeated until a fix point is reached: none of the reduction functions can decrease any of the domains further.

Definition 1 *For a p-ary constraint c an interval reduction function, r_c is of $p+1$ variables, which maps closed interval $p+1$ tuples to closed interval $p+1$ tuples. When given the intervals $I_1, \dots I_p I_{p+1}$ the reduction function returns the reduced intervals $I'_1, \dots I'_p, I'_{p+1}$ such that: $I_k \supseteq I'_k$ for $k = 1, \dots p+1$, and all the roots of f_c are within $I'_1 \times \cdots \times I'_{p+1}$.*

Note that there can be several reduction functions associated with a constraint, which differ in their strength, that is, how much of the non-solution is chopped off from the ends of intervals. In our case we use the inverse reduction functions, which provide the projection of the solution set within the given box on each variable, as defined below:

Definition 2 *The inverse reduction function associated with a p-ary constraint is an interval reduction function such that for the given intervals $I_1, \dots I_p I_{p+1}$ it returns the reduced intervals $I'_1, \dots I'_p, I'_{p+1}$ such that $I'_k = f_c^{-1}(I'_1, \dots I'_{k-1}, I'_{k+1}, \dots I'_{p+1})$ for $k = 1, \dots p+1$.*



a) b)

Figure 6: a) The solution set is the intersection of the box gained as the product of the domains and the region of all tuples satisfying the constraint. b) The bounding box around the solution set is drawn. Its projections onto the x- and y-axes are the reduced domains for x and y.

EXAMPLE 3 *Let us consider the binary constraint $\underline{I} \leq \frac{y}{x} \leq \overline{I}$, where $0 \leq \underline{I} \leq +\infty$, and the domain of the variables are $[\underline{x}, \overline{x}]$ and $[\underline{y}, \overline{y}]$, where $0 \leq \underline{x} < \overline{x} < +\infty$, and $0 \leq \underline{y} < \overline{y} < +\infty$. The allowed tuples are the ones within the box gained as the product of the domains, while the tuples fulfilling the constraint are the ones within the intersection of two half-planes, the solutions are the tuples in the intersection of the two regions. The reduced domains are defined by the bounding box of the solution set, see Figure 6. The reduced intervals for the variables can be explicitly given, by analysing the position of the corners of the box formed by the product of the initial domains, and the vertices of the trapezoid closed by the lines $\frac{y}{x} = \underline{I}$, $\frac{y}{x} = \overline{I}$, $x = \underline{x}, x = \overline{x}$, as given in Figure 7. If any of the lines $\frac{y}{x} = \underline{I}$ and $\frac{y}{x} = \overline{I}$ does not intersect the rectangle defined by the product of the reduced variable domains, then the domain for the constraint can be reduced too.*

For the rest of the basic animation constraint types the inverse reduction functions are similarly easy to compute. The invers reduction function produces tight endpoints for the reduced intervals, similar to the idea of box-consistency (Lhomme 1993).

We use the generic framework of *chaotic iteration* to describe our algorithm. This framework makes it easy to define variants, both because of the abstract level of specification and of the known theorems on convergence and completeness (Apt 1998). Below we give the iterative process to reduce the domains of an interval CSP.

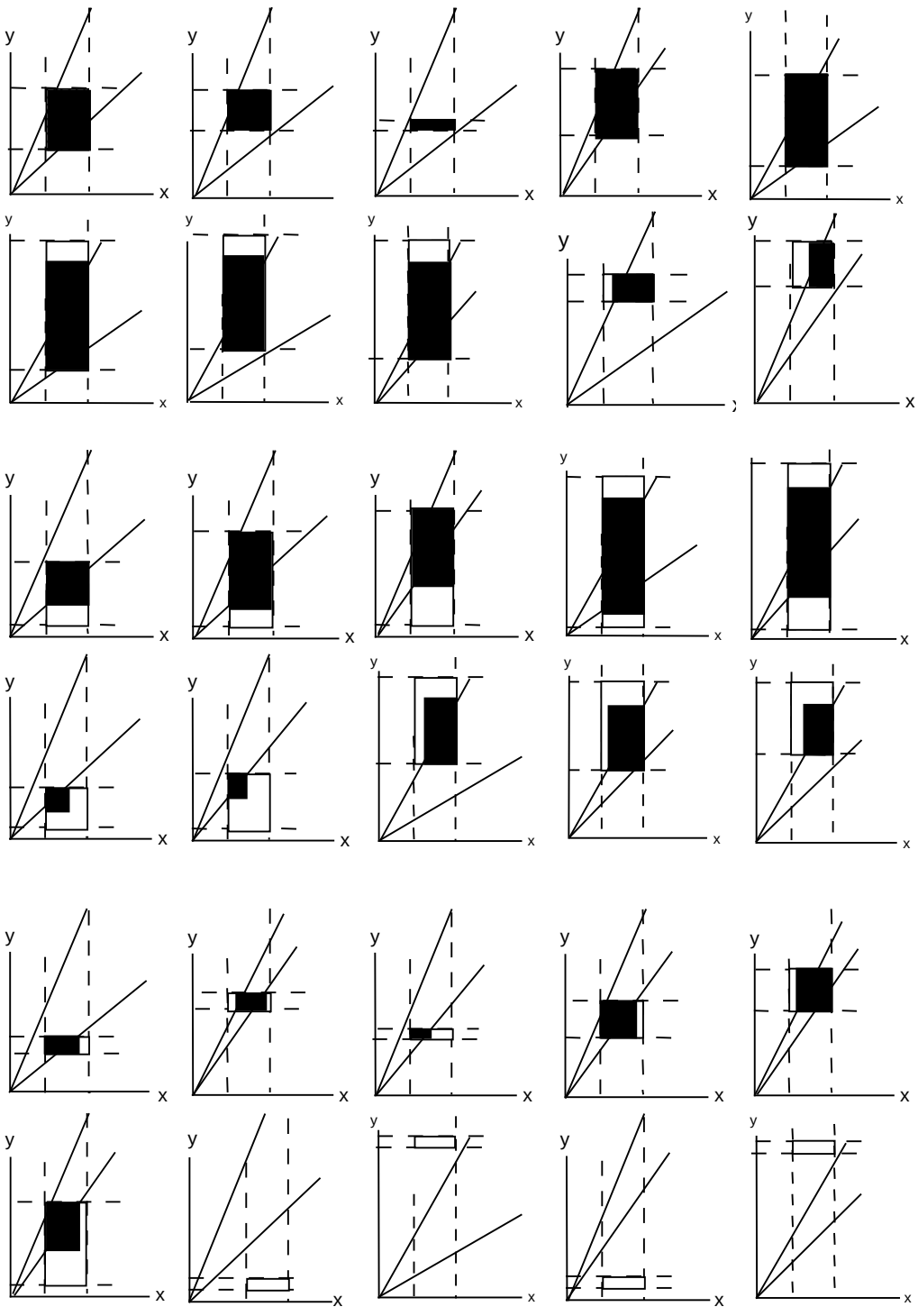


Figure 7: The possibilities for reducing the initial domains by the inverse reduction function for the constraint $\underline{I} \leq \frac{y}{x} \leq \overline{I}$, where $0 \leq \underline{I} \leq +\infty$. The initial domians are white, the reduced ones are black rectangles. In the top row the reduced domains are identical to the initial domains, while in the bottom row all but one of the reduced domains are empty.

```
1.  procedure Reduce_Domains(D, C*, C)
2.  current_domains ← D
3.  constraints_to_be_considered ← C*
4.  while constraints_to_be_considered ≠ ∅
5.      c ← Pick_One(constraint_to_be_considered)
6.      reduced_domains ← Reduce (c, current_domains)
7.      for all x in vars(c)
8.          if reduced_domains(x) ≠ current_domains(x)
9.          for all c' in C (c'≠ c and c' not in constraints_to_be_considered)
10.             if x in vars(c') then add c' to constraints_to_be_considered
11.             endif
12.         endfor
13.         current_domains ← reduced_domains
14.         endif
15.     endfor
16. endwhile
17. return current_domains
18. end
```

In our application we use finite discretized intervals of $[p, p+q, p+2q, ...p+nq]$ where p and q are rationals and n is integer. This ensures the termination of the iteration, and also that we need not worry about the usual problem related to precision of interval endpoints due to representational inaccuracy.

3.3 Generating a solution

The above procedure can be used interwoven with variable instantiations, to generate a solution by specifying a value and propagating its effect on the rest of the domains.

```
1.  procedure Find_Solution(D, C)
2.  solution ← ∅
3.  current_domains ← Reduce_Domains(D, C, C)
4.  if all domains in current_domains are non-empty
5.      constraints_to_be_considered ← ∅
6.      variables_to_be_instantiated ← vars(C)
7.      while variables_to_be_instantiated ≠ ∅
8.          instantiate all variables with a single domain and remove them from
9.                      variables_to_be_instantiated
10.         x ← Pick_One(variable_to_be_considered)
11.         v ← Pick_One(current_domains(x))
12.         current_domains(x)=[v, v]
13.         solution = solution ∪ ¡x,v¿
14.         for all constraints c (c not in constraints_to_be_considered)
15.             if x in vars(c) then add c to constraints_to_be_considered
16.             endif
17.         endfor
18.         current_domains ← Reduce_Domains(current_domains,
19.                     constraints_to_be_considered, C)
20.     endwhile
21. endif
22. return solution
23. end
```

Note that it is known after the termination of **Reduce_Domains**(D, C, C) if the problem has a solution or not. If there is a solution, then by propagating instantiated values, the current domains can be reduced further and no dead-end may occur. In the procedures **Reduce_Domains** and **Find_Solution** specific selection criteria can be applied instead of the general **Pick_One** selection. The above procedure can also be used to check if a partial instantiation can be extended to a solution, and if so, to generate an extension.

3.4 Generating default and random expression actions

An expression is defined as a (solvable) CSP. Whenever an expression action has to be produced, a partial instantiation of the variables of the expression has to be extended to a solution of the corresponding CSP. Typically, only the start-time of the expression action is specified. In this case, the partial instantiation with one time variable instantiated can be extended in many ways to a complete solution. For each expression, a method to generate a default solution — the default expression action — is defined. The default expression generation method extends the given variables to a default solution, using the **Find_Solution** procedure with deterministic built-in choices for instantiating remaining variables. However, often one just would like to have a variety of expression actions for the same expression. In such a case a random solution can be generated, on the basis of the above **Find_Solution** procedure, where choices for variable and value selection are made randomly.

3.5 Maintaining feasibility of the animation

When the user edits the animation, he changes the value of one or more variables of the animation, adds/removes CPs or modifies the set of requirements. As a result, the animation is not necessarily good any more, and the editor has to perturb the latest animation data to a new one which is a solution of the current constraints. The mechanisms taking care of the repair should be based on clear principles:

- to limit the **acceptable** amount of the repair,
- to choose the **best** one from all possible acceptable repairs.

Given a complete instantiation, the acceptable solutions are those solutions that can be gained by changing only a subset of the variables of the given instantiation. The non-changeable variables, the so-called *blocking variables* are identified dynamically, depending on the current animation, the change initiated by the user and some criteria on limiting the effects of the user's action. Typically, when changing an animation, one would prefer local effects. Hence variables of CPs far away (in time) from the CP being changed will be blocking variables. Also, the animator may prefer to work 'from left to right' in time. In such a case all CPs to the left from the currently changed CP should remain unchanged. If the timing of the animation should not be changed, then all time variables will be blocking ones. There are several further alternatives for defining the set of blocking variables. One set may contain others, but the ordering induced by inclusion is, in general, only partial. The user has the freedom to choose from the predefined alternatives, and switch from one to another to dynamically control the range of acceptable modifications.

In compliance with the propagation framework and the required fast response, the comparison of the possible repairs is based on an ordering of the variables and the difference between the old and new value. In the **Find_Solution** procedure, when a best solution is to be provided, instead of the **Pick_One** procedures for variable and value selection, some **Pick_Most_Important_Variable** and **Pick_Best_Value** procedures are used.

Repair in response to changes in variable values

The variables of CPs manipulated directly by the user are the *set variables*. The variables whith a value determined by the constraints, that is their domain contains a single value after domain reduction has been performed, are the *frozen variables*. In each situation the *free variables* are the

non-frozen, non-set and non-blocking ones. A repair is possible if after having propagated the value of the set and blocking variables, none of the variables have an empty domain.

As soon as a CP is grabbed, then a few CPs are identified as time or value blocking CPs. The value of the blocking variables is propagated, possibly resulting in a reduced interval for the time and value of the grabbed CP. The grabbed CP can be moved only within this reduced interval. When the grabbed CP is released, the effect of the change is propagated to the other variables.

Editing operations that involve a selection, that is CPs within a time interval, are considered as dragging all the involved CPs to their new location simultaneously, and the above-described approach is adapted.

Repair in response to manipulating the requirements

Changes of requirements can take place on three levels, by:

I. tightening/adding an individual one-time constraint;

II. changing constraints of one or more channels, with or without keeping constraints due to expressions;

III. modifying definition of expressions.

In case of level (I) changes, the blocking variables are all the ones outside the time region of the newly constrained CPs. Checking of feasibility of the added constraint and repair is done in a similar way as for editing multiple CPs.

In case of level (II) changes all the constraints which may be changed involve parameter variables. Repair is made by trying to preserve the timing of the animation to be repaired. There are several, partially ordered criteria to choose from to define the set of blocking variables.

The changes for a parameter may be in conflict with the definition of expressions. The option of discarding of expression definitions removes the constraints originating from the definition of expressions. The issue of removal of constraints is addressed in the next section.

With changes of requirements on level (III), the intention is either to replace certain expressions with others from the existing expression, or to redefine a facial expression and update the existing expressions accordingly. The change of expression may require the loosening/removing of certain constraints and tightening/adding others. The effect of loosening/removal of constraints is computed by re-doing the constraint propagation for the entire effected subproblem. The effected subproblem contains all the constraints which are connected to at least one loosened constraint. The added/tightened constraints are then propagated, taking into account blocking variables. For generating the best repair, different orders of CPs within the expression can be given which defines the order of instantiating the free variables within the expressions. The individual expression actions are taken from left to right.

Repair in response to adding and removing CPs

When the user initiates the insertion of a single CP, it is checked if there are requirements that prescribe the addition of constraints referring to the CP being inserted, and if those constraints are satisfied (see Figure 8.). If so, insertion takes place, and the effect of the removed/inserted constraints is propagated. Single CPs cannot be added to/removed from expressions.

If multiple CPs are to be inserted, similar checks take place for all the CPs. A piece of constrained animation can also be inserted. Then the constraints defining the inserted piece are also added.

3.6 Implementation

A first version of the parameter curve editor has been implemented in Java. This editor produces the 'scores' of the animation (see Figure 9.). A number of parallel lines — a 'staff' — are presented for each channel, and the control points (corresponding to musical notes) are to be placed in the staves. The effect of changes to scores can be seen directly, as the corresponding deformation of the face to

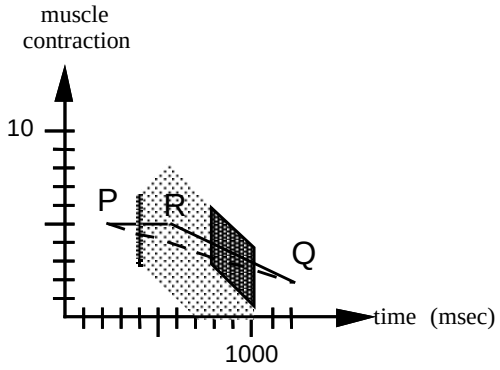


Figure 8: Two requirements limit the region for inserting a new CP: neighboring CPs should be no closer than 2 seconds, and the speed of the parameter value should be between -1 and 1. The region (given in light gray) where a CP can be inserted between the CPs P and Q. After inserting R, the feasible region for inserting a further CP has shrunk to two smaller regions (given in dark gray). Note that between P and R it is possible to insert a new CP only at time 400.

be animated is shown. Parts of the scores can be selected, and the resulting animation can be played to evaluate the visual effect.

The editor allows the insertion/deletion/dragging of single or multiple selected CPs. A selection can be copied and pasted, and scaled along time and parameter values. The user can select and manipulate pieces of more than one parameter curve at the same time, and perform the previous operations on all of them. Hence, it is possible to scale linearly an entire action, insert a smile and make repetitive blinks. Moreover, copy and paste is supported between multiple channels of different animations using different parameter profiles, as long as the number of channels is the same in the source and in the target. Appropriate bi-linear scaling takes place, making sure that neutral, minimum and maximum values correspond to each other in the channels copied from and to. Animations can be saved and read in as Java objects or ascii scripts, hence a library of pieces of animations to be reused as building blocks can be built up. This makes it possible to share pieces of animations between different facial models to be animated, and also to drive a synthetic face by — possibly edited — parameter curves gained by capturing the facial motion of a real human performer.

Only two most straightforward and general constraints are implemented, namely that the time of the control points should be increasing, and that parameter values should be within the domain given for the parameter. The manipulation of control points either individually or by performing operations on groups of them is revised automatically in order not to violate these constraints. There is no possibility yet to define building blocks in terms of constraints, but the editing and reuse of pieces of animation is supported.

Currently the new, fully constrained-based version is being implemented, with the above-discussed functionalities, also in the object-oriented style of the Java language. The new version will be extended with menu-based editing facilities to manipulate requirements and to choose for preferences for repair. The final choice for identifying blocking variables and alternatives for repair strategies will be limited, on the basis of how appropriate the effect of the different possibilities is in typical animation sessions. Also, the time of response to user's action purely based on interval propagation will be critically tested. For the final version it may be necessary to use generic or application-dependent heuristics for the order of applying reduction functions. The constraint-based definition of some basic expressions will be provided, allowing the generation of default and random expression actions and expression actions with different intensity. Besides the parameter channel level of visualization and editing, a level

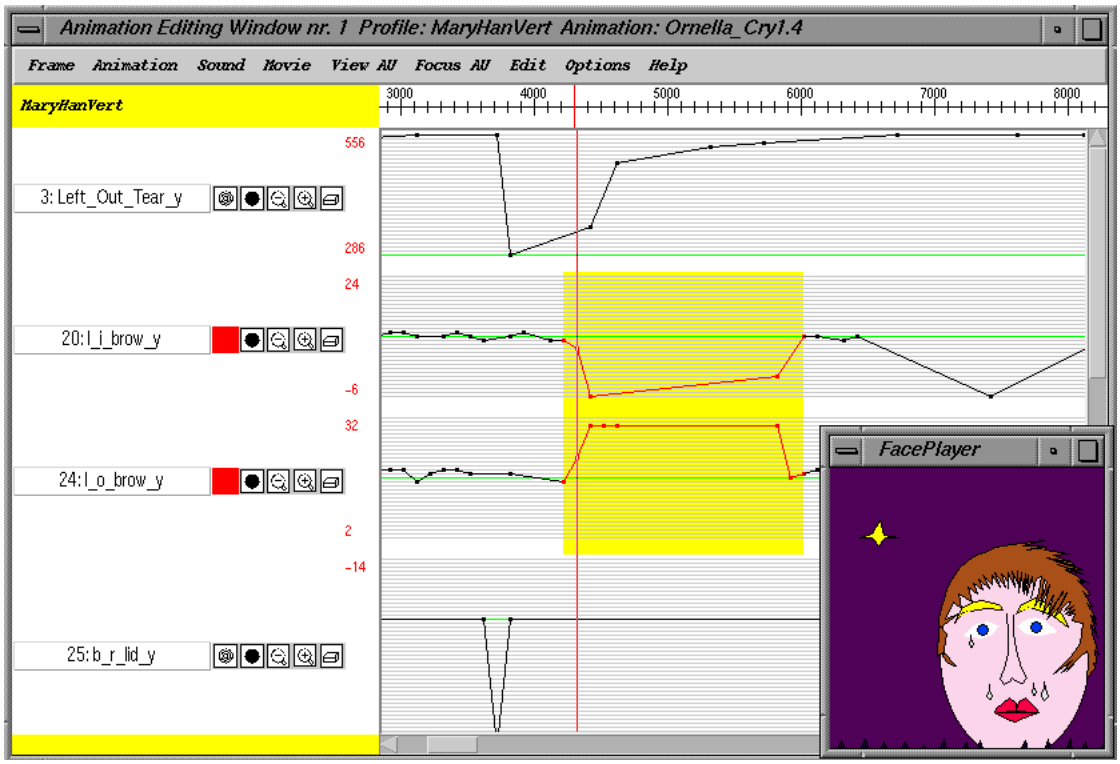


Figure 9: The Animation Editor, showing staves with data gained from captured facial motion and staves with synthetic, edited data. The second and third staves contain performer data to control the shape of the eyebrow. Data for crying and blinking were added by the animator. The top staff contains the parameter curve for dropping a tear, while the bottom one the curve for closing an eyelid. The shown cartoon face corresponds to the snapshot of the parameters at time 4300 msec. The highlighted portion of the animation, namely the eyebrow curves between 4200 and 6000 msec are selected. This selection can be edited: cut and copied, shifted in four directions, scaled, etc.

for expressions will be provided with a separate staff. Also searching for occurrences of expressions as well as replacement definition will be supported. Some visual feedback will be given on (effect of) constraints, showing frozen, blocking and free CPs in different colors. By clicking on CPs, the constraints referring to the CP will be listed.

4. Discussion

4.1 Further issues

As mentioned earlier, the solution method is not incremental with respect to **removal of constraints**. We hope that the time for recomputing the ranges of the effected variables will not be prohibitive, because of the sparse and loosely connected — though big — constraint graph. An other approach could be to keep track of the tightening of lower-upper bounds for variables similarly as done for interval arc-consistency algorithms (Cervone et al. 1994) and use this information to identify which ranges must be recomputed. For recomputation, some kind of 'resetting' propagation (Georget et al. 1999) could be used.

There has been little said about allowing the user to **see and directly manipulate constraints** set for an animation, and helping him to understand their effect. As long as the number of type of constraints is small, and constraints refer to variables of at most two CPs, the different constraints could be visualized as annotated curves connecting CPs, and the visual representation could be edited by direct manipulation. Allowing to visualize constraints within a time interval, and/or of certain source and scope would make such a visualization really helpful.

A facility for the **interactive definition of building blocks** — in contrast to writing the piece of code for the object and its methods — would be a good further extension.

Blending and concatenation of actions raises different types of questions. How to present and manipulate a piece of curve which is some kind of sum (Witkin et al. 1995) of two ordinary ones? Editing of blended actions should happen by editing one component at a time. Another type of question is if such a protocol is appropriate for the purposes of the animator — thinking in terms of components rather than the total effect. Moreover, setting requirements on the total effect is beyond of our framework, as would require some mechanism to reason about (sum of) curves in regions between CPs.

It has been often stated that computer animations look synthetic, because of the repetition of exactly the same, canned motion (Perlin 1995). The generation of random solutions and expression actions is our remedy. Another and more commonly used possibility is to add some **noise** to parameter curves. Shaking could be done at the stage of sampling the precise animation, adding noise to the parameter values per frame. It is an interesting question, if for facial animation some principles of 'good shaking' could be formulated.

4.2 Conclusions

We have presented an interactive graphical editor to be used for defining requirements for the facial movement to be produced and for composing facial animations which fulfil the requirements. The basic idea is that the requirements concerning the animation to be produced as well as the characteristic dynamical expressions and facial motion idiosyncrasies of the character can be expressed as constraints, and the concrete animation should be always a solution of the resulting CSP. The set of constraints to be satisfied are not known in advance, as the animator has the freedom to modify requirements interwoven with editing the animation. Moreover, the addition/deletion of CPs implies changes in the set of constraints. As the editor allows to declare and maintain requirements concerning an entire animation and reusable building blocks, it can be used as a 'motion sculpturing' tool. This is a novel functionality, in contrast to the single, concrete motion editing supported by other animation and motion synthesis tools.

The presented methodology can be applied for animation domains where there are no obvious and unique given constraints to relate the motion parameters of components, either because they do not exist/are not known, or the animator just wants to generate deformations and animation effects

beyond physical reality. Typically, cartoon character animation is such a domain.

From the point of view of constraint satisfaction, the task is to constantly repair the latest solution as a response to changes initiated by the user. The range of possible repairs — a subset of all the solutions — should be restricted dynamically. If this set turns out to be empty, the action initiated by the user is not carried out. Otherwise, the best of the possible repairs is selected and the animation is updated accordingly. The major service of the editor is to assure that the animator 'remains in the feasible region'. This is achieved by assuming that the feasible time and parameter range for each CP is a closed interval, and using interval propagation to recompute these intervals. Allowing certain types of monotone numerical constraints, the ranges are really intervals and can be computed fast.

Different principles for restricting the acceptable repairs and for comparing solutions can be defined by the user and incorporated into the general framework of interval propagation.

Acknowledgments

We thank Eric Monfroy for the useful discussions on interval propagation, Han Noot and Mark Savenije for implementing FaceEditor and Persona, and Paul ten Hagen for his comments on earlier versions of the paper. We are indebted for the remarks by the anonymous referees and for stylistic suggestions by Scott Marshall and Kálmán Ruttkay. The work has been carried out as part of the ongoing FASE project, sponsored by STW under nr. CWI 66.4088.

References

Alias Wavefront (1998) *Alias 8 Online documentation*, http://www.fh-jena.de/aliasguide/

Apt, K. (1998) The essence of constraint propagation, *CWI Quarterly*, Vol. 11. Nr. 2-3. pp. 215-249.

Benhamou, F., Older, W., Van Henteryck, P. (1994). CLP(intervals) revisited, *Proc. of International Symposium on Logic Programming (ILPS-94)*, pp. 124-138.

Benhamou, F., Older, W. (1997) Applying interval arithmetic to real, integer and boolean constraints, *The Journal of Logic Programming*, Vol. 32. Nr. 1. pp. 1-24.

Borning, A., Freeman-Benson, B. (1995) The OTI constraint solver: a constraint library for constructing interactive graphical user interfaces, *Proc. of the First International Conference on Principles and Practice of Constraint Programming*, pp. 624-628.

Borning, A., Anderson, R., Freeman-Benson, B. (1996) Indigo: A local propagation algorithm for inequality constraints, *Proc. of the ACM Symposium on User Interface Software and Technology*, pp. 129-136.

Borning, A., Freeman-Benson, B. (1998) Ultraviolet: A constraint satisfaction algorithm for interactive graphics, *Constraints*, Vol. 3., 1. pp. 9-32.

Brennan, S. (1985) Caricature Generator: The dynamic exaggeration of faces by computer, *LEONARDO*, Vol. 18. Nr. 3. pp. 170-178.

Bruderlin, A., Williams, L. (1995) Motion signal processing, *Proc. of SIGGRAPH'95*, pp. 97-104.

Cervone, R., Cesta, A., Oddi, A. (1994) Managing temporal constraint networks, *Proc. of the Second Int. Conference on Artificial Intelligence Planning Systems*, pp. 13-18.

CharToon Home Page (1998) http://www.cwi.nl/FASE/Cartoon/

Cohen, M. (1992) Interactive spacetime control for animation, *Proc. of SIGGRAPH'92*, pp. 293-302.

Da Silva, F., Velho, L., Cavalcanti, P. (1997) A new interface paradigm for motion capture based animation systems, *Proc. of Computer Animation and Simulation'97 Eurographics Workshop*, pp. 19-36.

Ekman, P., Friesen, W. (1978) *Facial Action Coding System*, Consulting Psychology Press Inc. Palo Alto, California

Essa, I. (1994) *Analysis, Interpretation, and Synthesis of Facial Expressions*. PhD thesis, MIT Medial Laboratory, available as MIT Media Lab Perceptual Computing Techreport #272 from http://www-white.media.mit.edu/vismod/

Essa, I., Basu, S., Darrel, T, Pentland, A. (1996) Modeling, tracking and interactive animation of faces and heads using input from video, *Proc. of Computer Animation'96*, pp. 68-79.

FaceWorks (1998) *DIGITAL FaceWorks Animation Creation Guide*, Digital
FAMOUS Home Page (1998) http://www.famoustech.com/
FASE Project Home Page (1998) http://www.cwi.nl/FASE/Project/
Georget, Y., Codognet, P., Rossi, F. (1999) Constraint retraction in CLP(FD): Formal framework and performance results, *Constraints*, Vol. 4. Nr. 1. pp. 5-42.
Gleicher, M., Litwinowicz, P. (1996) Constraint-based motion adaptation, *Apple TR* 96-153.
Guenter, B., Grimm, C., Wood, D., Malvar, H., Pighin, F. (1998) Making faces, *Proc. of SIGGRAPH'98*, pp. 55-66.
Hodgins, J., Wooten, W. L., Borgan, D. C., O'Brien, J. F. (1995) Animating human athletics, *Proc. of SIGGRAPH'95*, pp. 71-78.
Kokkevis, E., Metaxas, D., Badler, N. (1996) User-controlled physics-based animation for articulated figures, *Proc. of Computer Animation'96*, pp. 16-25.
Litwinowicz, P.C. (1991) Inkwell: a 2 1/2-D animation system, *Computer Graphics*, Vol. 25. Nr. 4. pp. 113-122.
Lhomme, O. (1993) Consistency techniques for numeric CSPs, *Proc. of IJCAI'93*, pp. 232-238.
Oster, G., Kusalik, J. A. (1998) ICOLA — Incremental constraint-based graphics for visualisation, *Constraint*s, Vol. 3. Nr. 1. pp. 33-59.
Owen, M., Willis, P. (1994) Modelling and interpolating cartoon characters, *Proc. of Computer Animation '94*, pp. 148-155.
Pachet, F., Delerue, O. (1998) MidiSpace: A Temporal Constraint-Based Music Spatializer, *Proc. of ACM Multimedia '98*, pp. 351-359.
Pachet, F. (1999) Constraints and musical harmonization: a survey, *Constraints*, to appear.
Parke, F., Waters, K. (1996) *Computer Facial Animation*, A K Peters.
Perlin, K. (1995) Real time responsive animation with personality, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 1. Nr. 1. pp. 5-15.
Persona Home Page (1998) http://www.cwi.nl/FASE/Spring/
Thórisson, K. (1996) ToonFace: A system for creating and animating interactive cartoon faces, *MIT Media Laboratory Technical Report*, 96-01.
Terzopoulos, D., Waters, K. (1993) Analysis and synthesis of facial image sequences using physical and anatomical models, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 15. Nr. 6. pp. 569-579.
Van Reeth, F. (1996) Integrating 2 1/2-D computer animation techniques for supporting traditional animation, *Proc. of Computer Animation'96*, pp. 118-125.
Van Hentenryck, P. (1989) *Constraint Satisfaction in Logic Programming*, MIT Press, Cambridge, MA, 1989.
Van Hentenryck, P. (1997) *Numerica*, MIT Press, Cambridge, MA,
Van Hentenryck, P., Laurent, M., Benhamou, F. (1998) Newton - Constraint programming over nonlinear constraints, *Science of Computer Programming*, Vol. 30. Nr. 1-2. pp. 83-118.
Williams, L. (1990) Performance-driven facial animation, *Proc. of SIGGRAPH'90*, pp. 235-242.
Witkin, A., Welch, W. (1990) Fast animation and control of nonrigid structures, *Proc. of SIGGRAPH'90*, pp. 43-252.
Witkin, A., Popovic, Z. (1995) Motion warping, *Proc. of SIGGRAPH'95*, pp. 105-108.
Vander Zanden, B., Myers, B. (1995) Demonstrational and constraint-based techniques for pictorially specifying application, objects and behaviors, *ACM Transactions on Computer Human Interaction*, Vol. 2. Nr. 4. pp. 308-356.