To appear in "Constraints in Artistic Applications" Eurographics'98 workshop proc.

Constraint-based keyframing

Zsófia Ruttkay, Paul ten Hagen, Han Noot

CWI Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
e-mail: {zsofi, paulh, han}@cwi.nl

1. Introduction: make a smile!

In this paper we will show the potentials of an animation editor to produce keyframes and inbetweening according to requirements of different origin and with different scope, all expressed in the form of constraints. The design of such an editor has been motivated by the particular task of producing realistic 3D [Par96] and cartoon-like 2D facial animations [Bre85], therefore we will use facial animation as the working example. However, the idea can be applied to other animation domains as well.

Computer facial animation has been a flourishing research topic for more than 25 years, aiming at building models 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 easily and intuitively deformed, and complying with the deformations of the real human face.

The so-called physically-based models [Ter93] 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, such as 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. Naturally, such 'minor' problems as the proper parametrisation of the individual muscles, the elasticity characteristics of the tissue layers, confirmation 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 faithful synthesised model of a given human face.

In parallel with the emergence of improved models, one has been confronted with the fact that there is neither enough knowledge on the dynamism of the human face, nor appropriate paradigms and tools to animate a face. 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 real muscles involved in facial expressions, and our task is to make this face smile. In order to simplify our discussion, let us 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 a 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 Fig.1. (It has been shown experimentally that the actual shape is far more complex [Ess96], but because of the lack of enough evidence on the real shape, trapezoid-shaped functions have been widely used.)

One can define infinitely many pairs of trapezoid-shaped muscle actuation functions - which ones produce an acceptable smile? How short or long can a smile be? 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.)

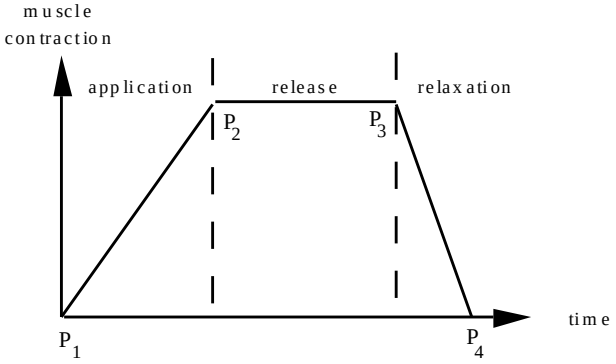


Fig.1.
Stages of muscle contraction

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)?

One may conclude that these questions are to be answered by analysing a huge sample of real smiles, and, moreover, with the development of face motion catching techniques a faithful animation could be and should be done on a performer-driven basis [Wil90].

There is research going on to accomplish the first task[Ekm78, Ess96], however, the problem seems to be very hard to tackle: it is practically difficult to get enough real, spontaneous facial expression samples recorded among 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 synthesised 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 creating animations for real or artificial faces. It would provide a basis to manipulate building blocks at different levels of abstraction and to ensure that the animation meets certain requirements (e.g. it is in line with the facial repertoire of a real person, without using captured data). Such a tool thus would make the process of creating animations easier and faster.

The other, even more exciting usage of such a tool is to define and experiment with characteristics of animations, e.g. the facial repertoire of the face to be animated, local and global synchronisation. One would use the editor as a tool to 'sculpture' the dynamism of a face first, and then make several animations according to the characteristics. Also, when editing a particular animation, the user would have tools to declare and modify requirements, and see the direct and indirect effect of these on the animation.

Though we will be using the smile as an example, on should remember that in reality the animator has to orchestrate possibly dozens of parameters, each for 20-30 frames per second. In spite of the size, the dynamism and complexity of the problems to be

handled, the editor should respond in real time, the underlying mechanisms should be transparent and easy to tune for the user.

In this paper we investigate how the above challenges can be met by the paradigm of constraint-based editing. First we discuss the representation of an animation and the requirements, and illustrate how the different animation editing tasks and actions can be expressed in terms of constraint processing. Then the technical questions of how to meet the constraint processing requirements will be dealt with. Finally, illustrative examples of the first experimental implementation are given and a plan for a complete implementation is discussed, and some open issues are raised. We finish with comparing our approach to alternative ones in animation editing and motion control for animation, and sum up the merits of constraint-based keyframing.

2. Constraint-based keyframing

2.1 The representation

We assume that an animation is defined by a given number N of parameters, each parameter taking its value from an interval of reals. (In this paper we do not address the technical issues related to discrete domains.) Hence, the evolvement of the animation along time is described by the vector of functions $(F_1(t),...F_N(t))$, where F_i: T-->D_i gives the value of the i-th *parameter* at the time t. The 0-th parameter is optional and is reserved to place time markers for synchronisation to absolute time.

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 keyframing and inbetweening [Whi88]. For most of our discussion, we will assume that the missing parameter values are computed by applying piecewise *linear interpolation* on the intervals between the time moments with given parameter values. In spite of this, we will use the notion *parameter curves* for the graph of the parameter functions.

Hence, for the i-th parameter channel, a number of $P^i_j = (t^i_j, v^i_j)$ *control points* (CPs) are given which define the animation. The number of control points may be different for different channels, and control points need not be aligned along time. If for a channel no control points are given, then the value of the parameter is assumed to be a default value (corresponding to the value of the parameter at neutral expression, in general).

The process of defining an animation requires to specify a sufficient number of control points, at proper times with proper values. The 'proper times and values', in general, means that the resulting $F_1(t),...F_N(t)$ functions together produce the requested animation. Many of the requirements concerning the animation to be produced can be well formulated and expressed as constraints on some of the co-ordinates of certain control points. The constraints all provide upper and/or lower limits on time or value of a control point, on time durations or changes of values in a time interval, on proportion of two time durations or of a time duration and the change of value during the duration. We will use extended intervals to indicate these limits: $I = [\underline{I} , \bar{I}]$ is a finite or infinite interval, that is $\underline{I}$ and $\bar{I}$ are reals, $-\infty$ or $+\infty$, and $\underline{I} \leq \bar{I}$. Defining the $\leq$ relation for the extended reals, this notation allows inequalities and equalities to be expressed in the form of membership in an extended interval. E.g. $x \leq 20$ will be expressed as $x \in [-\infty , 20]$. The type of constraints to be used are listed in Table 1.

When making an animation, it is common practice to re-use some earlier made pieces - such as a smile, a blink, visemes - as building blocks. These building blocks are given, just as any animation, by control points and constraints on some of the control points. We will refer to such building blocks as *actions*. We will use the same notion - e.g. a smile - both for the conceptual definition (which parameters are involved, what control points, what constraints) and for any instance of it (any set of control points fulfilling the constraints). For each action , there is a generic instance (e.g. a generic smile) which is used as default, if no preference is given for a specific instance.

(Ia)	$t^i_j \in I$	time range	(Ib)	$v^i_j \in I$	value range
(IIa)	$t^i_j - t^n_m \in I$	time duration	(IIb)	$v^i_j - v^n_m \in I$	value change
(III)	$(t^i_{j+1} - t^i_j) / (t^i_{k+1} - t^i_k) \in I$		relative time duration		
(IV)	$v^i_j / v^n_m \in I$		relative value		
(V)	$(v^i_{j+1} - v^i_j) / (t^i_{j+1} - t^i_j) \in I$		value change speed		

Table 1
The type of constraints to be used.
(I stands for a different interval in each case.)

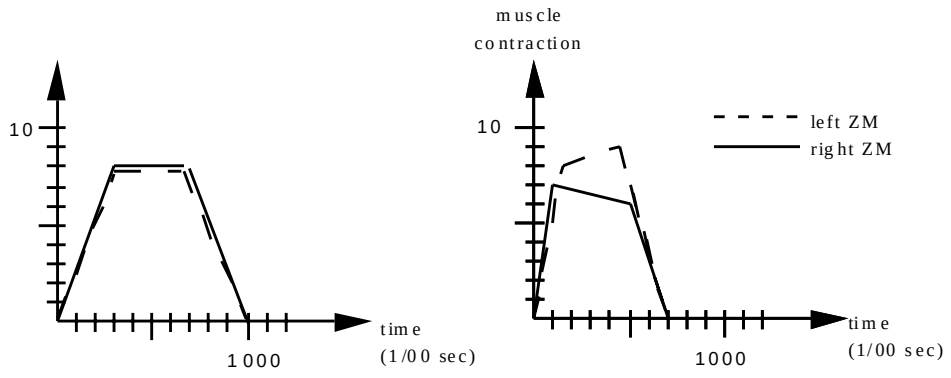
(0a)	$t^i_j \in [0, 1000]$	(0b)	$v^i_j \in [0, 10]$
(1a)	$t^1_2 - t^1_1 \in [50, 300]$	(1b)	$t^2_2 - t^2_1 \in [50, 300]$
(2a)	$t^1_3 - t^1_2 \in [100, 400]$	(2b)	$t^2_3 - t^2_2 \in [100, 400]$
(3a)	$t^1_4 - t^1_3 \in [100, 300]$	(3b)	$t^2_4 - t^2_3 \in [100, 400]$
(4a)	$t^1_1 - t^2_1 \in [0, 0]$	(4b)	$t^1_4 - t^2_4 \in [0, 0]$
(5a)	$t^1_2 - t^2_2 \in [-50, 50]$	(5a)	$t^1_3 - t^2_3 \in [-50, 50]$
(6a)	$v^1_1 \in [0, 0]$	(6b)	$v^2_1 \in [0, 0]$
(7a)	$v^1_4 \in [0, 0]$	(7b)	$v^2_4 \in [0, 0]$
(8a)	$v^1_2 \in [7, 10]$	(8b)	$v^2_2 \in [7, 10]$
(9a)	$v^1_3 - v^1_2 \in [-1, 1]$	(9b)	$v^2_3 - v^2_2 \in [-1, 1]$
(10)	$v^1_3 - v^1_4 \in [5, 8]$		

Table 2
Constraints of the smile action.
(t^1_j, v^1_j) are the control points defining the contraction function for the right,
(t^2_j, v^2_j) for the left Zygomatic major muscle.

We keep the notion *expression* for certain static configurations. Hence, the smile action refers to the dynamic process of smiling, while the smile expression is a frozen face with a smile. Similarly to actions, expressions are defined by the value of (some of the) parameters and eventual constraints.

In Table 2, the constraints defining one kind of smile action are given. Only unary constraints (type I) and binary constraints of type II are used. The unary constraints (0a) and (0b) restrict the domain of the variables involved, the binary constraints (1a,b), (2a,b), (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 synchronised, 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 unary constraints (6a,b)-(8a,b) restrict the domain of the control point values, while (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 actuation of one of the muscles.

The above set of linear equations has many solutions, each corresponding to a smile action. In Figure 2 the parameter curves for two smile instances are shown.



a b

Figure 2
Two specific smiles.

2.2 Constraints in different roles

Before addressing the technical question of dealing with sets of equations of the above types, we discuss the possible scope and origin of the constraints, and their different roles in the process of editing an animation.

Scope

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 parameters
  Examples:
  Limit on the length of the animation (Ia type).
  Limit on the time density of control points (IIa type). (Knowing the sampling frame rate of the animation, it makes no sense to try to specify anything within one frame, hence at most one CP should be given for one frame).

• *one parameter*
The constraint should hold for the entire animation for one parameter
Examples:
Limit on domain (type Ib).
Limit on speed of change of parameter (corresponding to speed of muscle contraction) (type IV).

• *one action*
The constraints should hold for control points of an action.
Example:
See definition of smile in Table 2.

• *certain parameters*
Throughout the entire animation or during a part of it, some parameters are coupled.
Example:
Complete or partial symmetry of the motion of the left and right face (type IIa,b).

• *local*
'One-time' constraint for a selected set of control points.
Example:
Synchronisation: two actions should begin at the same time (Ia, IIa), in a certain time moment the face should be smiling (Ib).

One should note that strictly speaking all the scopes but the last prescribe that instances of a certain kind of constraints should be added for certain variables (e.g. all the neighbouring ones for a given parameter, or all the value variables of control points which are within a time interval). We will not discuss further in this paper how, technically, this is to be done.

Source

When working on an animation, the constraints to be taken into account are of different origins. This is expressed by their *source*, which is one of the followings:

• the '*physical limitations'* of the face to be animated
Example:
Muscle contraction (speed and value) limits (IV).
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 taken into account.

• the *'behavioural repertoire'* of the character to be animated
Examples:
Asymmetric eyebrow movement, as typical for the character (IIa,b).
Specific smile (limits on timing and intensity).

• the *storyboard* of the animation
Example:
The facial expressions and lip-synch should be produced for a recorded text to be spoken by the character.

• the *animator*
Example:
The animator may add further, global or local constraints e.g. for synchronisation, may switch on-off symmetry, refine building blocks to be adapted, etc.

2.3 Constraint processing while editing

Editing an animation takes place by applying sequences of two kinds of primitive editing operations:
• *adding/deleting* (groups of) CPs;
• *changing value and/or time* of (groups of) CPs.

The above operations can be carried out by directly manipulating a graphical representation of the parameter functions via its control points. Interwoven with the manipulation of control points, the animator also changes the set of constraints in an *implicit* or *explicit* way:

- 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;
- constraints may be added/deleted/modified in a direct way, also on the level of requirements of all possible scopes.

The challenge is to provide an editing environment with the following services:

- the current animation is always feasible with respect to the current set of constraints;
- the feasible region is transparent for the user and can be well presented, and it is easy for him to remain in the feasible region while manipulating the animation;
- if as a result of a move by the user the current animation has to be adjusted to maintain feasibility, the 'best' solution should be found from the set of all possible ones;
- the user should have mechanism to influence the strategy for conflict resolution (define what is 'best' for him at the given moment);
- the solution method should be fast enough to provide real-time repair of values and to update feasible regions;
- some mechanisms should be available to point out which constraints are responsible for limitations on time and value of a CP;
- when adding constraints implicitly as a side effect of adding a CP, the animation should satisfy the new constraints as well;
- when adding/changing constraints explicitly, the animation should be updated to satisfy the new set of constraints automatically, again, providing the 'best' solution;
- tools should be provided to define and manipulate constraints on different levels (for individual explicit constraints, for requirements with different scope and origin).

Below, we illustrate in a few editing situations the above services.

At a certain moment of editing the animator decides to insert a smile. From all the possible smiles, the generic one (given in Fig 2a) is inserted, by inserting the corresponding 8 control points, and all the constraints on them. Then the animator wants to reshape the smile, by dragging the control point P^1_3 =(700, 8), the end of the release of the right ZM. The system knows (has computed) that there are no solutions for the set of constraints unless $t^1_3 \in [150,700]$ and $v^1_3 \in [6,8]$. Hence, the user cannot drag the point up (which would result in a value larger than 8) or right (which would result in a time larger than 700). The allowed positions are the points within a rectangle which may be highlighted to inform the user about possibilities. On request, the system can point out which time constraints cannot be satisfied outside the rectangle.

If the new position for P^1_3 is (700,7), then all the constraints remain satisfied. But if the user drags P^1_3 to (700,6), then the constraint (9a) is violated. This constraint can be satisfied again by one way only, setting P^1_2 to (300,7). With this change no other constraint

gets violated, so the process stops. If the user drags P^1_3 further to (500,6), two constraints get violated, (3a) and (5a). It is obvious that at least t^1_4 and t^2_3 have to be changed. Now it is up to the system to find out what is the best way to re-establish consistency. The default strategy may be based on the assumption that the animator would like to see as little further change as possible, and changes should be local. But the animator should be allowed to chose another criteria, such as that the new, repaired curve should have a similar shape as the original one.

The user may find that he needs a particular smile, outside of the feasible region defined by the standard set of constraints for a smile action. He wants to loosen some of the constraints, and go further with editing in the extended range of smiles. He may also add new CPs and some constraints to refine the shape of the parameter curves, e.g. to produce a more refined but still monotone application shape. He may need this modified smile more often later, so he may wish to save the definition of his new version of the smile. This does not mean saving the state of the corresponding control points, but rather, the set of constraints.

He may want not only to change constraints individually, but several ones for an entire time interval or for an entire parameter, by manipulating requirements and possibly modify their scope, resulting in modification of all the involved constraints and similar repair as in the case of modifying an individual constraint.

When tightening constraints or adding new ones, the animator should be prevented from introducing conflicting constraints (resulting in an empty set of animations as solutions). However, he should be given some information about the conflicting subset of constraints which prohibits his action.

3. The constraint solver

In general, to solve a CSP is difficult. Fast and complete algorithms exist only for special types of problems. Moreover, the additional requirements (listed in 2.3) such as to cope with dynamical changes in the set of constraints and with underconstrained problems, and to compute all feasible values per parameter limit the choice of solution methods to be applied. Note moreover, that we are dealing with continuous domains, so the corresponding problem has usually infinitely many solutions. So the following scenario makes sense:

(i) produce an interval for each parameter containing all feasible values;
(the solution set is contained in the box defined by the product of these intervals),
(ii) select the 'best' solution from the infinitely many possible ones on the basis of the current selection criteria.

This scenario is suitable from the user's point of view, as:
• it gives information on the range of feasible solutions;
• it allows him to use different criteria to prefer a particular solution.

Because of the different nature of the ordering constraints of type I and II and the rest, two different solution methods (both of the above style) can be used. If there are only constraints of type I and II, the advantages of a special and well investigated algorithm can be exploited. If other types are present as well, a general interval propagation method can be used. The common characteristics of both methods is that they provide a strict, minimal interval of feasible values for each variable.

3. 1 Ordering constraints only

Constraints of type (I) and (II) prescribe some ordering of the time or the value of control points. Problems built up of such types of constraints are well-known as a special case of temporal constraint satisfaction problems, the so-called *simple temporal problem* (STP) [Dec91]. In our case, actually, we have two independent STPs, one for the co-ordination along time, and one for the shape of the parameter curves. These two problems can be solved independently, and often, only one of the two problems is affected by a change due to the user.

STPs are known to be not NP-hard, in contrast to more general variants (e.g. one with a choice of alternative constraints between variables). For our purposes the best is to use the specialised AC-3 algorithm discussed in [Cer94], which solves the problem by establishing arc-consistency. The algorithm is linear and thus can produce a solution fast. Moreover, the produced arc-consistent equivalent problem is decomposable, that is, taking any partial instantiation of the variables satisfying the constraints between them, this partial solution can be extended to a complete solution. Hence, variables can be (re-)instantiated in any order, and without backtracking.
The algorithm also keeps track of which constraint was used last to narrow the range of a variable in the process of interval propagation, which allows to detect quickly if a new constraint to be inserted or an old to be tightened would cause inconsistency, and to update the feasible intervals if a constraint is relaxed/removed.

3.2 The general case

The nice characteristics of the STP do not remain valid if constraints of type (III)-(V) are also present. In this case, the general techniques used to solve interval constraints can be applied [Ben94]. The idea is to narrow down the set of possible values for variables via some kind of arc-consistency fixed-point iterations. The method proposed by Lhomme [Lho93] propagates bounds on projections of the constraints similarly to the AC-3 like algorithm used for STPs, with the essential requirements that the min-max values of the projected constraints can be easily computed to be used for narrowing the domain of other variables effected by the same constraint. The process stops either by producing a 'box' envelop of the solution set with strict boundaries, or detects that no solution exists. (Eventual problems of looping can be handled.) In our case the constraints are all linear, thus the feasible values for an individual variable is an (may be infinite or empty) interval. Hence the approximation provided by the enveloping box will produce exactly the interval of feasible values for each variable.

Individual solutions can be approximated by splitting and narrowing the feasible regions per variable. To find a 'best' specific solution takes place by adding further constraints expressing the selection criteria. This tighter problem can be solved incrementally, starting from the current intervals for the variables. So quickly either a solution is produced or the tighter problem is proven to be inconsistent, and the set of constraints for the selection criteria have to be revised. Though there is little known (to our knowledge) on theoretical estimates on performance, in practice the response time has turned out to be acceptably short [Ben94, Ben97]. Thus it seems to be realistic to use such a technique for selection strategies which can be expressed in terms of (explicit or implicit) ordering of sets of constraints to be used to tighten the problem.

By keeping track of the effect of narrowing on the individual variables, it is possible to point out the constraints which were responsible for determining the narrowest interval of feasible values per variable. However, in general, this information is not sufficient to relax intervals when removing constraints.

The main point where this method is inferior to the special one applicable for STPs is how the change of one variable effects others. Now a change in a single variable may cause, indirectly, changes of variables of satisfied constraints as well. Hence, the animator cannot fix the value of more than one variable, in principle. Note, however, that as the sub-problems effected by one basic editing operation are usually small and the appropriate subproblem is often loosely constrained, in practice one can still expect that freezing variables not related to directly changed ones works. As the solution methods is fast, at least the 'as little change as possible' principle can be applied.

3.3 Possible extensions to the set of constraints

A big advantage of the general interval constraint method is that it allows the extension of the type of constraints as well as the structure of the problem. The types listed in Table 1 will likely turn out to be too restrictive, depending on the application domain. E. g. for realistic smiles, one would probably need linearly approximated exponential curves. For cartoon-like animations one would need trigonometric functions to be able to define easily e.g. movement of the pupil along an ellipse.

Another useful extension would be to be able to express conditional requirements in the form of disjunctions of constraints, e.g. to express that if the mouth is very much open then many of the muscles around the mouth cannot be contracted. Such conditions, declaring that two different sets of constraints are to be considered in two split parts of an interval of a variable domain, could be well handled in the framework of interval propagation.

5. Discussion

5.1 Present state and plan for incremental implementation

At present, a very first version of the parameter curve editor is available, implemented in Java. When using this editor, the user is writing a 'score' of the animation (see Fig.3). He is presented with a number of parallel lines - a staff -for each channel, and the control points (corresponding to musical notes) are to be placed in the 'staff'. He can scroll along time and along the parameters, hide currently not to be edited staves. The editor allows the basic editing operations on control points and scaling and shifting along time and parameter values.

There is no possibility yet to define building blocks in terms of constraints, but the editing and re-use of pieces of animation is supported. 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 an entire smile, insert a smile, make repetitive blinks. Moreover, cut and paste is supported between multiple channels of different animations using different parameter profiles, as long as the number of copied channels is the same as the number of ones in the target. When pasting a piece of curve to a channel with ranges and neutral value different from those of the original one, appropriate linear scaling takes place on the intervals above and below the neutral value. This makes it possible to share pieces of animations between different models to be animated, and also to drive a synthetic character by - possibly edited - parameter curves gained by capturing the facial motion of a performer. Animations can be saved for later re-use as Java objects, hence a library of pieces of animations to be re-used as building blocks can be built up.

Only two most straightforward and general constraints are implemented, namely that the time of the control points should be monotonously increasing, and that values

should be from the prescribed domains. The manipulation of control points either individually or by performing operations on groups of them is restricted automatically in order not to violate these constraints. The value constraints play a role when scaling selections and when at cut and paste the two parameter channels have a different profile. There is also an option to allow only discrete parameter values of different granularity.

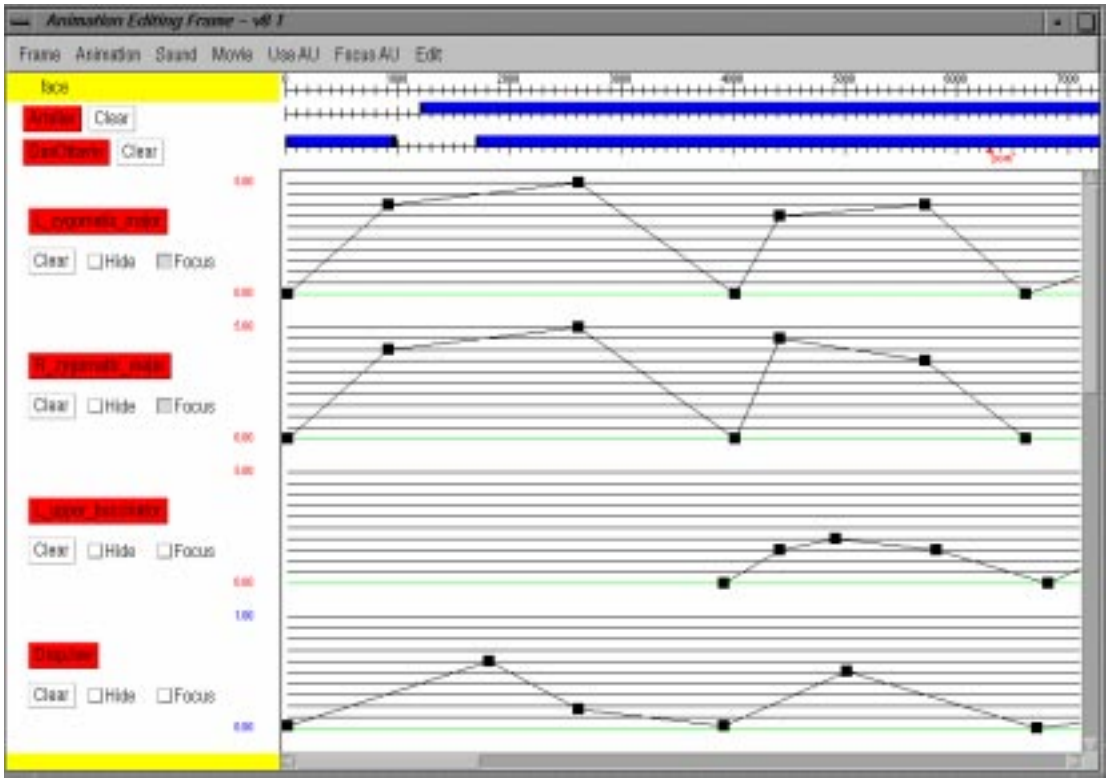


Figure 3
Snapshot of the editor
There are 2 audio channels and 4 parameter staves shown.

Sound arrangement can be given as well. Audio channels can be added above the staves for the animation parameters, and the playing instructions for each channel can be given by placing start/suspend/continue/stop symbols at proper time moments in the line for the audio channel. Moreover, if pre-set labels are available for the audio channel (they can be obtained by several MIDI tools, we use the SGI tool soundtrack) , then these labels are shown in the corresponding playing interval, and can be used as markers to synchronise the animation.

The present version is meant to be a try-out of the user interface concept and also as a simple tool to be used to make animations. We use the editor in two environments: to animate a 3D physically-based generic face model, and to animate 2D cartoon-like faces. For the latter purpose, the editor is used as a component of the CharToon package we developed [Noo98]. The advantage of the 2D animation is that animations can be directly played in acceptable quality (with a the non-optimised animation player, 20 frames/sec on an empty Sparc station for complex faces with dozens of control parameters). Secondly, the effect of the control parameters is more straightforward than in the case of the complex physically-based model. Finally, in the CharToon environment one can design faces with different parameter profiles and experiment with re-using and transforming pieces of animation made for different faces.

Several users with different background should use the system with the CharToon environment in a 3 months try-out period, hopefully resulting in valuable comments for the next versions. Namely, interviewing the users we expect feedback on:
• the type of constraints to be used;
• the type of strategies to be used to provide a solution for an underconstrained problem;
• preferences for paradigms to visualise and manipulate constraints;
• what other than linear interpolation should be supported.

Meanwhile, the constraint solvers explained earlier will be implemented. A special, though important issue is if this can be done with success in Java. Knowing of an interval propagation algorithm implementation already in Java[Bra98] and of the continuous improvements in the Java language versions, we hope to be able to do so, keeping in mind the wide range of potential (test)users who would like to run the editor on very different machines. Also it will be investigated if characteristics of the specific types of constraints to be used finally (e.g. linear only) can be exploited. The algorithms in [Lho93] will be extended with keeping track of constraint dependencies, and with the capability to find a preferred solution. The editor will be extended with appropriate editing and visualisation tools for constraint manipulation, and ones to express preferences.

5.2. Further issues

The possibility to allow non-linear interpolation raises the question of the intention of constraining control points. One prescribes constraints on the control points in order to achieve some characteristics of the entire parameter curve. In case of linear interpolation, the types listed in Table 1 have clear and local consequences on the 'inbetweened' part as well. So only such interpolation should be allowed for which the constraints on control points imply constraints on the entire curve, and the way how the control points effect the parts inbetween, should remain transparent for the user. Hence, approximated curves are excluded, and possibly also higher-order interpolated ones too, unless continuity of the first derivate will turn out to be a must for animators.

Blending and concatenation of actions poses similar questions. How to present and manipulate a piece of curve which is some kind of sum (for possibilities, see [Wit95]) of two ordinary ones? The original control points will not be on the curve any more. One would like to keep the control points associated with the component curves, but then anomalies like two CPs at the same time may arise. A way out of this problem is to treat pieces of curves specially if they are the sum of ordinary ones. The editing operations (both of CPs and constraints) would apply only to control points, hence editing of blended actions should happen by editing one component at a time. For the time of editing a component, the piece of curve of that component with its control points should be drawn, and the effect of editing should be shown on both the components curve and on the blended result.

It has been often stated that computer animations look synthetic, because of the repetition of exactly the same motion [Per95]. The suggested remedy for this is to add some noise to parameter curves. This could be used in our case too, in two different ways. Either when the animator is done with composing the animation, a 'shake' operation would perturb the solution. The degree of perturbation could be influenced by declaring if some constraints could be violated to some extent. Shaking could be done not only for entire animations, but on pieces of the animation or on particular actions and parameters as well. It is an open question if 'shaken' animations could be further edited. As an other possibility, shaking could be done at the stage of sampling the precise animation, adding noise to the parameter values per frame. The result - playing the same animation would produce slightly different effect each time - may be a cheap solution to just what

is needed in some applications. It is an interesting question if for special models some principles of 'good shaking' could be formulated and used.

5.3 Comparison to other approaches

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. We believe that constraint-based keyframing is an appealing tool because it does not restrict the animator more than he needs that, as it is basically he who declares what constraints to include and where. This methodology can be applied for domains where there is no obvious and unique set of constraints to be applied, either because they are not known or unique (such as in the case of facial expressions) and/or specific constraints should be applied locally (e.g. to restrict the location of an object for a short time only). Hence, in our case, the manipulation of constraints is an essential part of the design and editing of an animation, and not used only to compute a solution (generate a proper animation). These three aspects make our approach different of other usage of constraints for animation.

In the case of systems applying dynamical constraints, [Hod95, Kok96, Wit90] these universal constraints express a piece of the physics behind the motion and deformation of real objects, while in the case of inverse kinematics [Wat92] general motion and geometry equations and constraints can be used. In both cases, the set of constraints can be changed by changing some 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 parameters (location, velocity) at certain times [Coh92]. In such cases, keyframing has a limited role.

On the other hand, many of the commercial packages do allow the manipulation of motion curves [Mae96]. However, manipulation is usually restricted to one channel at a time (corresponding to location and speed co-ordinates), and there are no means to define even simple constraints e.g. for synchronisation of channels.

Motion warping techniques [Wit95] and signal processing based motion curve transformations [Bru95] have similarities with our approach, namely that they are transforming a 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 in the case of constraints. The closest to our approach is the system outlined in [DaS97], which allows processing and re-use of curves obtained by motion capture only. The success of these techniques in spite of their limitations show the urging need for tools to manipulate and re-use pieces of animations.

Acknowledgement

We thank for Eric Monfroy for the useful discussions on interval propagation.
The work has been carried out in the framework of the ongoing FASE project [FAS98], supported by STW grant CWI66.4088.

References

[Ben94] Benhamou, F., McAllester, D., Van Hentenryck, P.: CLP(Intervals) revisited, Proceedings of International Logic Programming Seminar 1994: 124-138.

[Ben97] Benhamou, F., Older, W.: Applying interval arithmetic to real, integer and boolean constraints, The Journal of Logic Programming, 1997: 1-24

[Bra98] The Brandeis Interval Arithmetic Constraint Solver, http;//tigereye.cs.brandeis.edu/Applets/IAsolver.html

[Bre85] Brennan, S.: Caricature Generator: The dynamic exaggeration of faces by computer, LEONARDO, 18(3): 170-178

[Bru95] Bruderlin, A., Williams, L.: Motion signal processing, Proc. of SIGGRAPH'95: 97-104.

[Cer94] Cervone, R., Cesta, A., Oddi, A.: Managing temporal constraint networks, Proc. of the Second Int. Conference on Artificial Intelligence Planning Systems,1994, pp 13-18.

[Coh92] Cohen, M.: Interactive spacetime control for animation, Proc. of SIGGRAPH'92, Computer Graphics 26(3):293-302.

[DaS97] Da Silva, F., Velho, L., Cavalcanti, P.: A new interface paradigm for motion capture based animation systems, Proc. of Computer Animation and Simulation'97 Eurographics Workshop, 1997, pp 19-36.

[Dec91] Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks, Artificial Intelligence 49:61-95

[Ekm78] Ekman, P., Friesen, W.: Manual for the Facial Action Coding System, Consulting Psychologists Press, Inc., Palo Alto, CA, 1978.

[Ess96] Essa, I., Basu, S., Darrel, T, Pentlands, A.: Modeling, tracking and interactive animation of faces and heads using input from video, Proc. of Computer Animation'96:68-79.

[FAS98] FASE home page: http://www-it.et.tudelft.nl/FASE/

[Hod95] Hodgins, J., Wooten, W. L., Borgan, D. C., O'Brien, J. F.: Animating human athletics, Proc. of SIGGRAPH'95:71-78.

[Kok96] Kokkevis, E., Metaxas, D., Badler, N.: User-controlled physics-based animation for articulated figures, Proc. of Computer Animation'96:16-25.

[Lho93] Lhomme, O: Consistency techniques for numeric CSPs, IJCAI'93, pp 232-238.

{Mae96] Maesti, G.: Digital Character Animation, New Riders Publishing, Indianapolis, Indiana, 1996.

[Noo98] Noot, H., Ruttkay, Zs., Ten Hagen, P.: CharToon: A Java-based animation system, Submitted to Eurographics'98 as short paper.

[Par96] Parke, F., Waters, K.: Computer Facial Animation, Peters, Wellesley, Massachusetts, 1966.

[Per95] Perlin, K.: Real time responsive animation with personality, IEEETransactions on Visualization and Computer Graphics, 1(1):5-15.

[Ter93] Terzopoulos, D., Waters, K.: Analysis and synthesis of facial image sequences using physical and anatomical models, IEEE Trans. Pattern Analysis and Machine Intelligence, 15(6):569-579, June 1993.

[Wat92] Watt, A., Watt, M.: Advanced Animation and Rendering Techniques, ACM Press, New York, 1992

[Whi88] White, T.: The Animator's Workbook, Watson-Guptill Publications, New York, 1988.

[Wil90] Williams, L.: Performance-driven facial animation, Proc. of SIGGRAPH'90, Computer Graphics 24(3): 235-242.

[Wit90] Witkin, A., Welch, W.: Fast animation and control of nonrigid structures, Proc. of SIGGRAPH'90, Computer Graphics 24(3): 243-252.

[Wit95] Witkin, A., Popovic, Z.: Motion warping, Proc. of SIGGRAPH'95, pp 105-108.