InterFace: a Real Time Facial Animation System

José Daniel Ramos Wey
João Antonio Zuffo

LSI - Laboratório de Sistemas Integráveis
Escola Politecnica de Universidade de São Paulo
Av. Prof. Luciano Gualberto, 158 trav 3 Cidade Universitária, SP, Brazil

wey@lsi.usp.br
jazuffo@lsi.usp.br

Abstract. This paper describes InterFace, a real-time facial animation system. The system uses a very simple set of pre-modeled facial expressions to create a wide range of emotions, mouth positions and complete head movements. The animation is done through layered groups of actions that combined gives the virtual actor complete freedom to do different actions independently and simultaneously, without the intervention of the animator. Due to its software structure, InterFace can be used as a “render plug-in” for an artificial intelligence program or a script-based animation system. The system was successfully implemented using the languages Java and VRML, and can be executed through the Internet with PCs or workstations with a Java-enable Web browser.

Keywords: Computer Graphics, Real Time, Facial Animation, Facial Expressions, Morphing, Virtual Actors.

1. Introduction

Computer generated human characters is one of the most important areas in Computer Graphics lately. Agents or avatars are present in multimedia titles, graphic user interfaces, tele-presence and shared virtual worlds systems. While these agents can represent a breakthrough in human-computer interaction, it is not easy to create and animate human characters that work efficiently. When we are dealing with computer-generated agents, we turn on our human responses. We expect them to act like humans, moving the lips when they are talking, expressing emotions, having a personality. We expect human behavior from a computer program.

An essential propriety for adding a human personality in a computer generated character is the facial animation. Psychological researches show that humans express most of the communication and the emotions by the face [Mehrabian, Ferris (1967)]. However, creating believable human faces in a computer is not an easy task. We have a remarkable ability to recognize one face between thousands of similar faces, and we can detect very subtle changes in facial expressions [Parke, Waters (1996)]. It's correct to say that all humans are specialists in human faces.

The construction and animation of human face models in computers is not a new area. Since Parke’s first studies [F. Parke (1972)], several techniques were created to add realism to 3D computer human faces. However, animating a face convincingly in real-time is still a problem today.

The most used techniques to animate human faces are morphing between fixed and variable polygon topology [Parke (1972)][F. Parke (1974)][N. M. Thalmann et al. (1989)] and muscles simulations [K. Waters (1987)][C. Wang et al. (1994)]. While muscle simulations can give good results, it is computationally expensive and difficult to implement. Morphing between faces are easy to implement and computationally cheap, but usually requires the animator to model all the facial expressions and mouth positions before hand.

We propose a facial animation system that allows easy creation and animation of facial expressions. It is simple to implement and not too expensive computationally, being suitable with complete human simulators (with body movements, behavior, and so on). The facial expressions are created visually, based on a simple set of pre-modeled expressions that combined can lead to a wide range of facial emotions and mouth positions. The animation system use a layered approach, allowing easy combination of different actions, like saying a phrase while the character is angry. The animation of different actions is composed with a model similar to image composition.
2. System Overview

To develop a facial animation system, we have to consider at least two aspects: modeling and animation.

The model is the representation of the head. It can be done in different ways: modeling from photographs, 3D scanning or modeling using special software. Models can use different representations, like polygon mesh and B-splines. The animation can also be done in several different ways, like morphing between different pre-modeled faces, simulating the muscles of the face, or using sensors or trackers.

In order to create a real-time system, our approach uses polygon-based models and morphing between faces for animation. Instead of using muscle simulations, we decided to use a visual approach that is more intuitive to artists and animators.

The InterFace system has two modules: the expression modeler and the animation module. The expressions are modeled visually and interactively, and the results can be seen in real-time. The facial animation is done by composing groups of actions. An action is a bit of animation the virtual actor can perform, like crying, talking, blinking or moving the eyes. Actions of the same kind are joined together in groups. Each group of actions has a transparence value to be used when the actions are composed, in a way similar to image processing algorithms.

The InterFace system is based on a previous work done in the Improv System [Perlin (1996)] and the Aria project [Ruschionni et al. (1997)]. It is implemented using VRML [VRML (1997)] for the description of the scene and basic expressions, and Java [Java] for the animation and modeling modules. These languages were chosen because they are system independent and have a good performance. The InterFace system can be accessed online, in the URL http://www.lsi.usp.br/~wey/interface/. The system requires a Java compatible browser with a VRML 2.0 plug-in. The platform used for development was a Pentium PC with Windows 95 and Netscape Communicator.

3. Modeling facial expressions

The process of animating a face usually requires modeling all facial expressions and mouth positions by hand. When the scene is long or complex, this approach can easily become impractical - the number of expressions to be modeled is too large.

To solve this problem, Frederick Parke [F. Parke (1974)] used parametric models of the face. By controlling a small set of parameters (like jaw rotation, eyelid opening and mouth positions) it is possible to create different facial expressions easily. Each parameter controls a region of the face, allowing transformations like rotation, scaling, vertices position offsets and interpolation.

The InterFace system uses the same approach, but in a different way. Complex expressions are modeled combining a small set of pre-modeled expressions, called the Basic Library of Expressions (BLE) - it is very similar to what Frederick Parke called "parameters". This specific set of BLE was created based on the research done by Ken Perlin [Perlin (1997)]. This set is not based on the physical characteristics of the face. The objective is not to simulate the face muscles, but to allow a "cartoonish" character to express itself emotionally and synchronize lip movements with the voice. We adapted the BLE defined by Perlin to 3D, since Perlin's work was done in a 2D character. Of course, the system is not limited to the BLE. We can use other expressions if necessary. The BLE is just a good set of expressions for creating emotions and mouth positions.

The BLE is composed by the following expressions: eyes move up, eyes move left, rotation of the left eyebrow, rotation of the right eyebrow, blink left, blink right, lower eyelid move up left, lower eyelid move up right, sneer left, sneer right, smile left, smile right, mouth position like "ahh", mouth position like "ohh", head rotation in Z, Y and X axis. A screenshot of the InterFace expression modeler is shown in figure 1.

Each expression in the BLE is stored by the system as differences from the original face of the character (called the canonic expression). For instance, given the canonic face and the face with the mouth like "ahh", the system will calculate which vertices moved (in this case, only the vertices from the mouth) and store these differences as an array of vertices offsets and vertices indexes. The vertices that do not move are not stored. Beyond vertices offsets, the system can use translation, rotation and scaling of objects that compose the face.
The system modeler module presents 17 sliders, each one representing the weight of each expression of the basic library. While Park's implementation used linear interpolation from 0 to 1 for each parameter, our system uses a new approach: each expression has a weight on the model. If only one weight is set to 1 and all others are set to 0, there is no difference between this expression and the original expression from the basic library. Each weight can range from -∞ to +∞, although the useful values almost never exceed -5 to +5.

The final expression is composed by simply adding the differences stored for each expression in the basic library, multiplying by the respective weights. The order that the weighted expressions are added makes no difference in the final expression. Figure 2 shows an example of the result of adding three expressions.

Although the BLE is relatively small, we were able to model several different and interesting expressions. Some of them are shown in the figure 3.

We also modeled a small set of mouth positions that can be used efficiently for synchronizing the lip movements and the voice, as describe in [P. Blair (1989)] and [B. Robertson (1997)]. It is a set of eight mouth positions that mimics most of the phonemes of the English language, creating the illusion that the character is actually speaking. This set has been used for traditional cartoon animation for many years.
Figure 3: Some expressions modeled using the BLE

All expressions created in this module are stored as arrays of weights of BLE. The expression *LOOK_RIGHT* has weight 0 for all BLE except the value –1 in the “look to the left” basic library expression. The last expression shown in the image above is described by the following array of BLE weights: [0.2 -0.3 3.8 4.1 -0.3 0.4 -0.4 -0.7 0.1 0.6 0.2 -0.4 0.6 -0.2 0.1 0.1].

4. Animating the face

Once we have some expressions modeled in the module described above, it is time to animate our character. As stated before, the InterFace system is composed of two modules: the expression modeler and the animation engine.

The animation engine is based on the concept of layers of actions, developed by Ken Perlin in his recent work on virtual actors [Perlin (1994)][Perlin (1995)][Perlin (1996)]. We adapted the concept of layers of actions to solve the specific problem of facial animation.

![Figure 4: The Interface Animation module](image)

4.1 Actions

The basic element of the animation system is the Action. An action is a bit of animation the virtual actor can do. Examples of actions are talk, look to something, breath, sneeze, say a phrase, being angry, happy and so on.

Actions are created by controlling the weight of pre-modeled expressions with mathematical functions (called curves in our system to be more intuitive for animators). All the expressions modeled in the expression modeler are available to the animation engine. When the weight of an expression is set to one, it is the same of setting the array of weights of the BLE defined by this expression.
The curves available in the InterFace system are: constant, linear interpolation, B-splines, sine, cosine, non-predictable impulse and Perlin Noise [Perlin (1985)]. Since Java is an object-oriented language, it is very easy to add more mathematical functions as needed – just create another class inherited from the curve abstract class that implements the function.

The expression curve is involved by an envelope as shown below:

![Envelope Diagram](image)

Figure 5: How the envelope affects the curve

This envelope defines a time to start (where the weight of the expression is zero, no matter the value of the function), a time to “fade in” (where the value of the function is multiplied by a linear interpolation from 0 to 1), a “duration” time (where the value of the function is not changed) and a “fade out” time (where the value of the function is multiplied by a linear interpolation from 1 to 0). The duration time can be infinite. The curve values are not limited to [0,1], although most of time this range is used. Several expressions can be used in the same action. When two or more expressions are combined in the same action, the system just add the weights, therefore the order that the expressions are specified is not important.

The action itself is involved in an envelope. For each action we specify a time to fade in, a duration time (which can be infinite) and a time to fade out. These times are especially important in the transitions from one action to another.

The system uses a simple language to create actions. The syntax is:

```
ActionName Tin Tdur Tout
Expr1 Tst Tin Tdur Tout curve params
Expr2 Tst Tin Tdur Tout curve params
...
```

*Tin* is the time to fade in, *Tdur* is the duration of the action or the curve of the expression and *Tout* is the time to fade out. All times are given in seconds. The parameters of the curve are dependent to the type of curve used.

To illustrate how actions are done, we show below the code that defines the action *sneeze*:

```
Action Sneez 0.1 9.6 0.3
  rotx 0 0 10 0 spline 0 0 3.5 1 4 2 5
  -2 7 0
  blink 3 0 2.5 2 constant 1
  Ahhh 0 3 0.5 0.5 constant 1
  whistle 3.5 0.1 2 constant 1
  roty 7 0.2 1 0.2 noise 0.3 1
  rotz 7 0.2 1 0.2 noise 0.3 0.6
```

The action has a total time of 10 seconds (0.1 to fade in, 9.6 sustain and 0.3 to fade out). We defined in the expression modeler that the expression *rotx* rotates the head in about 30 degrees in the X axis. We used a B-spline curve to animate this expression: in time 0 seconds, the value of the weight for this expression is 0. From 0 to 3.5 seconds, the weight is smoothly interpolated from 0 to 1 (which is the movement for the “ahhh” phase in a sneeze). From 3.5 to 4 seconds, the weight is smoothly interpolated from 1 to –2, which is the movement for the “choo” in the sneeze. And so on. The *blink* expression closes the eyes during the sneeze. The *Ahhh* and *whistle* perform the mouth movements for “ahhh-choo” and the *roty* and *rotz* expressions gives an impression of itching the nose after the sneeze.

One of the key elements to add realism to the characters is the use of Perlin noise [Perlin (1985)]. In order to make actions like blinking and breathing realistic, the timing must be non-predictable, but it must be controlled as well, so the actor will not take too long to blink or blink too fast – these variations are handled very well with the noise function. As a result, the use of Perlin noise helps avoiding the impression of “mechanical” movements.

Actions can be interrupted at any time. When an action is interrupted, it automatically starts to “fade out”, unless, of course, the action is already in this state. The fade out time in actions is used to avoid rough transitions when an action is interrupted.

### 4.2 Groups of Actions

In order to improve performance and make the process of animation easier, actions of the same kind, like emotions, mouth movements and head movements, are grouped...
All actions that belong to a group are usually not performed at the same time, except during the transition from one action to another—where one action is “fading in”, while the other is “fading out”. Of course, actions from different groups can be performed simultaneously.

Sometimes it may be interesting to perform two different actions of the same group simultaneously. To make it possible, the system automatically creates a group that contains all the actions from previous groups, so actions that were grouped together can be performed in the original group and this one simultaneously.

While the system allows any action to be grouped together, we achieved the best results using the following groups: non-intentional actions (like breathing and blinking), emotions (like crying, smiling and sleeping), lip movements (the eight lip positions described above that mimics the phonemes), eyes movements and head movements.

Each group has a transparency or alpha value, ranging from 0 to 1. Groups with alpha set to 0 are completely transparent and with alpha set to 0 are completely opaque. The alpha value is used in the composition of actions, described below.

### 4.3 Composition of Actions

The virtual action can perform several actions at the same time. Each action that is been executed may contribute to the final scene, depending of the alpha value of the group it belongs. The model for composing the actions is similar to techniques used in image composition with transparency [Levoy (1998)].

In a given time, the system verifies in each group the actions that are being performed. If no action is performed in a group, it skips to the next. If an action is performed, probably this action is multiplying an array of modeled expressions by curves, resulting in an array of weights in the BLE. The system accumulates the weights of the BLE from group to group using the following algorithm:

\[
[W_{\text{scene}}] = [W_{\text{scene}}] \times \alpha_{\text{group}} + [W_{\text{group}}] \times (1 - \alpha_{\text{group}})
\]

Where \([W_{\text{scene}}]\) is the array of weights of the BLE of the scene, \([W_{\text{group}}]\) is the array of weights of the BLE calculated for the actions that are been performed by this group and \(\alpha_{\text{group}}\) is the transparency value of the group. Of course, groups with alpha value of 0 do not influence the final scene. The actions use transparency also: for instance, actions that only make movements on the lips are transparent to the rest of the face.

The order in which the groups are calculated is important! Changing the order can give a totally different result. Groups that are calculated later usually affect the scene more than groups calculated earlier. The group order can be changed on the system in real time.

This technique simplifies considerably the work of an animator: the virtual actor can perform actions like crying, talking, blinking and moving the eyes at the same time. The animator acts more like a director to the scene, coordinating what the virtual actor will do, instead of actually moving the objects and vertices.

### 5. Conclusions and Future Work

The InterFace system was designed to be used by animators or to be integrated with other systems. In order to stick to this goal, we decided to make the interface and the inputs to the system as simple and visual as possible. Although the actions are not create visually right now, actions created for one actor can be easily used in another, so libraries of actions can be made allowing the animation to be done very easily. The use of Java and VRML for development made it possible to share our program with users all over globe through the Internet, allowing an easy way to distribute the software and receive feedback from testers.

The InterFace system will be improved in different ways. We are working on a script language that can start and stop actions and change the group order and transparency. Scripts can be used to drive a scene directly or can be combined in libraries to be used as “building blocks” of larger scenes. Since InterFace is a real-time system, we can read and execute scripts in real-time. This way, any program that uses CG humans as its user interface (like programs that simulate human behavior, computer games, virtual agents or avatars) can use InterFace to render the facial expressions of the virtual actor.

In a near future we will add the ability of changing the color of the face, thus simulating vascular activity [P. Kala, N. M. Thalmann (1994)]. This would increase the realism of the virtual actor—when the actor is feeling fear, his face would become white, when he is angry or shy, his face become red and so on.

Finally, we are working on recreating the environment of the Aria project, presented in Siggraph 1996 [Ruschionni et al (1997)] on this system, allowing it to run...
in PCs through the Internet instead of graphical workstations.

6. Acknowledges
This work would never be possible without the help and suggestions of Ken Perlin, professor of The New York University. He is also responsible for the name InterFace. Arnaldo Massato Oka created the 3D model. Everyone involved in Aria also helped in the development of this project, especially Marcelo Zuffo and Ruggero Ruschioni. Daniel Wey would like also to thank Raquel Pires Gonçalves for the supported and for lending her animation books. This project was partially supported by the administration of the University of São Paulo.

7. References:
References are listed in order of appearance:


Java - http://www.javasoft.com/


Ken Perlin, *Danse Interactif*, Siggraph ’94 Electronic Theater, 1994


Ken Perlin, *An image synthesizer*, Siggraph ’85 proceedings, 1985
