ToonFace: A System for Creating and Animating Interactive Cartoon Faces

Kristinn R. Thórisson

Gesture and Narrative Language Group, Media Laboratory
Massachusetts Institute of Technology
20 Ames Street E15-410N Cambridge MA 02139
http://www.media.mit.edu/~kris
kris@media.mit.edu

Abstract. The majority of facial animation work has focused on realistic rendering, often with complex underlying models of muscles and sometimes of facial tissue. A system for facial animation is described that takes a simpler, more artistic aproach. A face animated in this system is an effective vehicle for emotional and communicative expression. The scheme emplyed divides the face into seven features, each with a specific number of control points. Control points are moved with one- and two-dimensional "motors." Three kinds of filled polygons can be used to define areas and lines in the face. The simplest kind is a static, non-animated polygon, the second kind is associated with a whole feature (and all its control points), the third kind is simply associated with a single control point. As the control points are moved, either in one or two dimensions, the shape and position of animated polygons is modified to conform to the change, and thus change the facial expression. To achieve this, two kinds of interpolations are used. A face in this system is semi-3-D: it is defined in a 2-D plane that can be turned in three dimensions to simulate head turning and nodding. The result is a flexible system that can generate a range of facial expressions. The system comes in two parts, one being an Editor for constructing faces, the other being an animation engine for animating a face in real-time.

Keywords: *Facial animation, caricature, real-time interaction, agents, human-computer interation.*

1. INTRODUCTION

While computer graphics work concerned with faces has to date focused extensively on visual appearance, interactivity and effectiveness for information transmission via the face has not been of primary concern. The motivation for the current work comes from an interest in the social metaphor for human-computer communication where one or more synthetic characters, often referred to as "agents," take the role of a participant in a face-to-face conversation [Thórisson 1994b, Nagao & Takeuchi 1994, Thórisson 1993, Laurel 1990]. As the modes of speech, gesture and gaze become a routine part of the computer interface [Koons et al. 1993, Bolt & Herranz 1992, Bolt

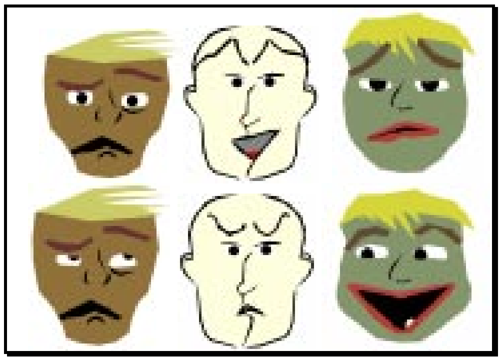

Figure 1. Examples of faces and expressions generated in ToonFace.

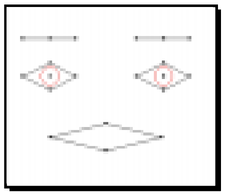

Figure 2. Seven objects comprise the animated parts of the face: Two eye brows, eyes and pupils, and one mouth. Control points (shown as dots) can be positioned anywhere within the face, by selecting and moving them with the mouse.

1980, Mochizuki et al. 1992, Thórisson et al. 1992, Britton 1991, Neal & Shapiro 1991, Tyler et al. 1991] the demand increases for effective facial displays on the computer's side that can facilitate such multi-modal interaction.

Making facial computer animation look convincing has proven to be a difficult task. A common limitation of physically-modeled faces [Essa 1995, Essa et al. 1994, Waters 1990, Takeuchi & Nagao 1993, Waite 1989] is that the meaning of their expressions is often vague. An ideal solution to this would be to exaggerate facial expression, but within a physical modeling framework this may look unconvincing or awkward. An alternative is what might be called a "caricature" approach [Thórisson 1993, 1994a, 1994b, Britton 1991, Laurel 1990] where details in the face are minimized and the important features therefore exaggerated (see Hamm [1967] for an excellent discussion on cartooning the head and face). In this fashion, Brennan [1985] created a system that could automatically generate caricature line-drawings of real people from examples that had been entered by hand. Librande [1992] describes a system called *Xspace* that can generate hundreds of artistically acceptable two-dimensional drawings from a small example base. Simplified faces seem like a very attractive alternative to physical modeling for animating interface agents, both in terms of computational cost and expressive power. This report describes a system called *ToonFace* that uses a simple scheme for generating effective facial animation along these lines (Figure 1). It differs from prior efforts primarily in its simplicity and its way of representing facial features. The following section is a more elaborate discussion of the motivation and goals behind this work. Section 3 describes the particulars of drawing and animation routines, section 4 gives a quick tutorial of the two parts of ToonFace, the Editor and the Animator. A short user guide follows in section 5 gives a short comparison between ToonFace and the Facial Action Coding Scheme (FACS, Ekman & Friesen 1978). Lastly, current applications and future enhancements are described in section 6.

2. MOTIVATION AND GOALS

Most current systems for facial animation are very complex, include between 70 and 80 control parameters [Essa 1995, Essa et al. 1994, Terzopoulos & Waters 1993, Waters & Terzopoulos 1991, Waters 1987] require powerful computers and seldom run in real-time. There is a clear need for a simple, yet versatile method of animation that allows for interactive control. ToonFace is an attempt to create such an animation package. The primary goal of ToonFace is to create facial expressions in real time in response to a human interacting with it. ToonFace meets this requirement by being simple: mostly two-dimensional graphics with five kinds of polygons (three of which are user-definable) and four kinds of polygon manipulations. It employs very simple linear interpolation methods for achieving the animation—a clear win under time-constraints. By reducing the degrees of freedom in the movements of the face to a managable number (21 df), it is easier to control of the face than in most other approaches. A

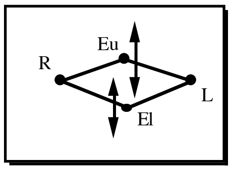

Figure 4. Each eye has four control points, but only two of those move. Upper (Eu) and lower (El) control points have one degree of freedom each in the vertical.

secondary goal of the system is that it meet mininal criteria for graphical quality and look. The scheme employed allows people to use their own artistic abilities to create the look that they need for their system.

3. ToonFace ARCHITECTURE

ToonFace consist of two parts, an *Editor* and an animation engine or *Animator*. The Editor allows a user to construct a face within a point-and-click environment. The Editor runs on an Apple Macintosh™ computer in Macintosh Common Lisp (MCL) [Macintosh Common Lisp Reference 1990, Steele 1990]. The Animator is a C/C++ program running on an SGI using OpenGL [Neider et al. 1993] routines for real-time rendering. We will now look at how a face is represented in ToonFace and the drawing and animation routines.

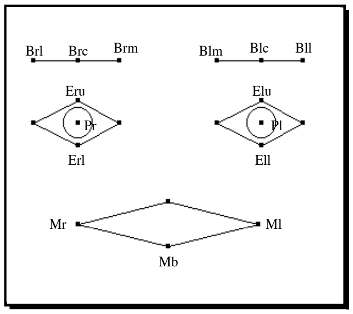

Figure 3. Codes used for the animated control points.

3.1 Facial Coding Scheme

A face is divided into seven main features: Two eye brows, two eyes, two pupils and a mouth. The eye brows have three control points each, the eyes and mouth four and pupils one each (Figure 2).

Control points that can be animated are given the codes shown in Figure 3. These points were selected to maximize the expressive/complexity tradeoff (see section 6). In the case of points that can move in two dimensions, each dimension is denoted as either “h” for horizontal or “v” for vertical (Figures 4, 5 & 6). The following is a complete list of all one-dimensional motors that can be manipulated in a face [control point number in brackets]:

```
Brl = brow/right/lateral [3]; Brc = brow/right/central [2]; Brm = brow/right/medial [1]
Bll = brow/left/lateral [6]; Blc = brow/left/central [5]; Blm = brow/left/medial [4]
Eru = eye/right/upper [7]; Erl = eye/right/lower [9]
Elu = eye/left/upper [8]; Ell = eye/left/lower [10]
Plh = pupil/right/horizontal [15]; Plv = pupil/left/vert [15]
Prh = pupil/right/horiz [16-h]; Prv = pupil/right/vert [16-v]
Mlh = mouth/left/horizontal [14-h]; Mlv = mouth/left/vertical [14-v]
Mrh = mouth/right/horizontal [13-h]; Mrv = mouth/right/vertical [13-v]
Mb = mouth/bottom [12]
Hh = head/horizontal [17-h]; Hv = head/vertical [17-v]
```

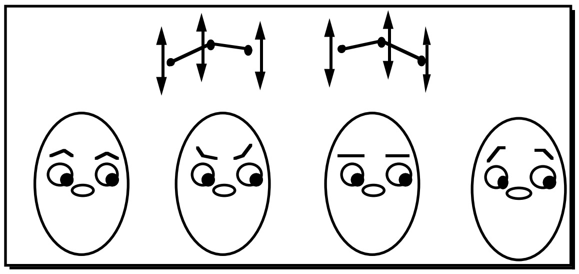

Figure 5. Eye brows have three control points, each with one degree of freedom in the vertical.

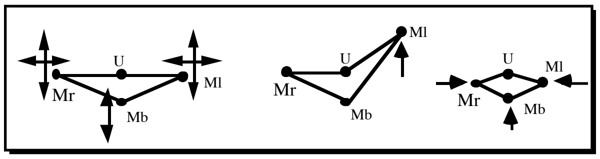

Figure 6. The mouth has four control points, three of which actually move. The ones on the sides (Ml & Mr) have two degrees of freedom, the bottom control point (Mb) has one.

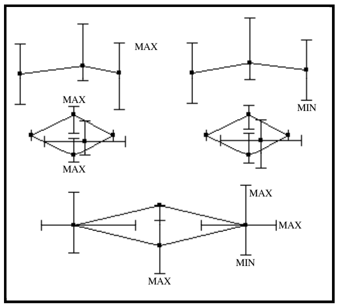

Figure 7. Limits of control point movement and direction of their dimensions.

Horizontal motion is coded as 0, vertical as 1. Each of the motors can move a control point between a minimum and a maximum position (for a given dimension). Thus, *max* and *min* values mark the limits of movement for each motor. For the eyes and head, these are given in degrees, (0,0) being straight out of the screen; upper left quadrant being (pos, pos), lower left quadrant being (pos, neg). Figure 7 shows these limits as they appear graphically in the Editor. A line extends the full range of a control point's path. The limits can be changed by clicking and dragging the ends of these lines.

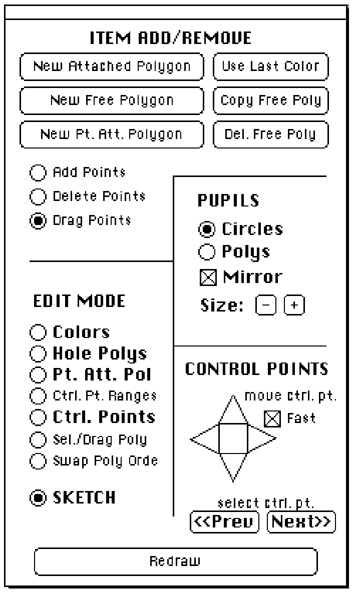

Figure 8. Control panel for the ToonFace Editor (see text). (Pt. Att. Pol = point attached polygons, Ctrl. Pt. = control points; Sel. = select).

3.2 Drawing Scheme: Polygons

As mentioned before, drawing is done by filled, two-dimensional polygons. There are three kinds of user-manipulable polygons which all can have an arbitrary number of vertices. A new polygon is created by selecting the desired type from a menu (Figure 8), then selecting the feature or control point to attach it to (unless it is a free polygon). A polygon is moved by dragging it; its vertices are changed by dragging them to the desired locations. A new polygon always has eight vertices, which can be deleted or added to as desired.

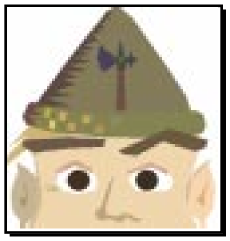

Figure 9. Free polygons are used for objects that don't have to move relative to others, like hats, hair, nose and ears.

Free Polygons.

This is the simplest kind of polygon in the system. Free polygons are simply drawn in place and cannot be animated. They are used for constructing features that do not need to move relative to other features, including hair, ears, decorations, scars, etc. An example is given in Figure 9.

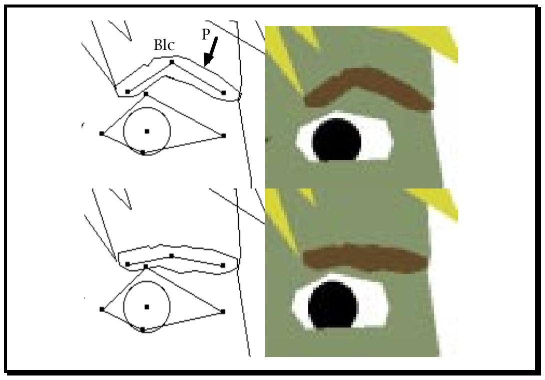

Figure 10. As the central control point on the left eye brow (Blc) is moved down, the vertices of its attached polygon (P) are recalculated according to how the angle of the lines between the control points changes. The left side shows the control points of the eye brow with connecting lines, the right side shows the polygons when filled.

Feature-attached Polygons.

These polygons are associated with a whole feature. An example is a polygon representing an eye brow (Figure 10). These polygons are animated in relation to the whole

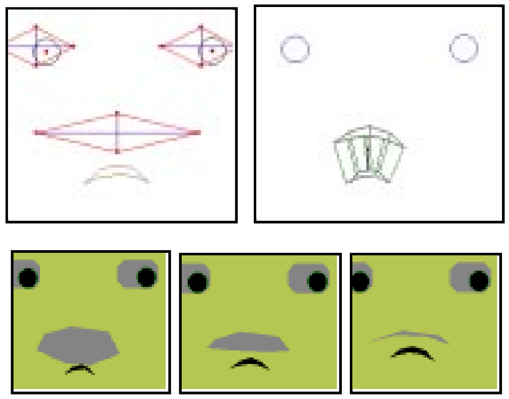

Figure 11. Polygons attached to a single control point have two defined states (shown in the upper right with lines connecting common vertices). As the control point moves (in this case the bottom mouth point), the vertices of the polygon are interpolated between the two pre-defined states.

feature: if one point in the feature moves, the points on the polygon closest to that point are recalculated and redrawn: as a result, the polygon changes shape.

Point-attached Polygons.

A point-attached polygon only changes form/position when a single control point—the point to which it is attached—changes position. The user defines two states for the polygons, one showing how it should look when its control point is at its *max* position, the other corresponding to its *min* value (Figure 11). When the control point is moved during animation, a linear interpolation is performed between the polygon's two states.

Drawing Order.

For purposes of making features overlap correctly, three kinds of special-case polygons are used. *Hole polygons, pupils* and the *face polygon*. Hole polygons are the insides of the eyes and mouth. When the face is drawn, the hole polygons are drawn first, then the pupils, then the face polygon—except for the regions defined by the hole polygons—then the free polygons, then point-attached polygons, and lastly the feature-attached polygons:

```
STEP
 1.    DRAW (HOLE POLYGONS)
 2.    DRAW (PUPIL POLYGONS)
 3.    DRAW (FACE POLYGON) — (AREAS DEFINED BY HOLE POLYGONS)
 4.    DRAW (FREE POLYGONS)
 5.    DRAW (POINT-ATTACHED POLYGONS)
 6.    DRAW (FEATURE-ATTACHED POLYGONS)
```

3.3 Interpolation Algorithms

The control points of a face's feature are connected by lines, as shown in figures 4, 5, 6 and 10. These lines are used to determine how the feature-attached polygon's vertices move when any single control point on the feature is moved. A feature like the left eyebrow has three control points (Bll, Blc, Blm) which all move in the vertical dimension. In Figure 12 h0 and h1 are the horizontal positions of Blm and Blc; the vertical would be {v0, v1}. From these the slope of L1 is determined:

$$S_l = (v1 - v0) / (h1 - h0)$$

The y-intercept of line L1 is given by:

$$Iyl = v0 - (S_l * h0)$$

The following method is then used to calculate the position {x,y} of a vertice v on a feature-attacghed polygon P:

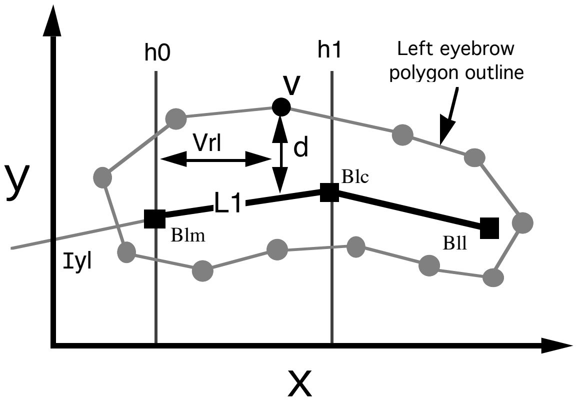

Figure 12. Example of polygon point interpolation (see text).

$P = \{v_1, v_2, v_3, \ldots\}$
$v = \{x, y\}$

$x = h0 + (vrl * (h1 - h0))$
$y = (x * S_l) + I_yl + d$

where vrl is the relative horizontal position of point v betweeen h0 and h1 (along line L1) and d is the distance of point v from L1. This is exemplified in Figure 10: When the control point Blc is moved down, vertices on polygon P move to keep a constant distance to the lines between the control points, resulting in a new shape for the eyebrow.)

The feature lines are not used for point-attached polygons. These simply have two states, one for the control point's max position, and another for its min position (Figure 11). The following linear interpolation method is used to calculate a point-attached polygon's vertice (v) value {x,y}:

$v = \{\text{min-x}, \text{min-y}, \text{max-x}, \text{max-y}\}$

$x = v_{\text{min-x}} + (P_{\text{ctrl}} * (v_{\text{max-x}} - v_{\text{min-x}}))$
$y = v_{\text{min-y}} + (P_{\text{ctrl}} * (v_{\text{max-y}} - v_{\text{min-y}}))$

where P_{ctrl} is the position of the associated control point along its min-max dimension (a float between 0.0 and 1.0).

3.4 Animation Scheduling Algorithms

The Animator part of ToonFace uses a multi-threaded scheduling algorithm to simulate parallel execution of motors. The main loop has a constant, loop-time, which determines the number of animation frames per second. The value for this constant should be equal to the maximum time the main loop could ever take to execute one loop. In the current implementation this constant is set to 100 ms, giving a fixed rate of 10 animation frames per second. When a command to move multiple motors is received, the total time this action is supposed to take is divided into loop-time slices. Since all motors are independent from each other, separate slices are made for each motor. So for a close-left-eye command (i.e. control point Elu) of a 500 ms duration, 5 slices would be made for the left eye, each slice to be executed on each main-loop. If the eye is fully open when the command is initially recieved, the eye will be 20% closer to being fully closed on each loop, and fully closed when the last slice has been executed. If a command for closing both eyes in 500 ms were to be given, a total of 10 slices would initially be produced and each time through main loop one slice for the left eyelid and one slice for the right eyelid would be executed, bringing both eyes to a close in 500 ms. If all pending slices have been executed before the 100 ms loop-time constant has been reached, the program waits the remaining time, thus guaranteeing a constant loop time.

Here is a rough outline of the main loop in pseudo-code:

```
LOOP FOREVER
    Start-Time = read-clock
    commands-received = Read Socket Input
    IF commands-received
        FOR each motor IN commands-received
            MAKE-SLICES
    FOR each motor
        EXECUTE-ONE-SLICE
    PAUSE (loop-time — (read-clock — Start-Time))
```

The faster the rendering, the lower the loop-time constant can be set, resulting in smoother animation. The value for this constant is most easily chosen by experimentation, since execution time of depends on various factors, such as number of slices in each loop, amount of commands received per second, etc., whose interactions are difficult to predict.

It is expected that the program connecting to the ToonFace Animator contain libraries of standard motions, such as smiling, frowning, neutral appearance, etc. This is a non-trivial issue and will not be discussed here.

4. A QUICK GUIDE TO USING ToonFace 1.0

4.1 The Editor

When ToonFace is loaded in Macintosh Common Lisp [Macintosh Common Lisp Reference 1990, Steele 1990], two windows open, and a menu appears. A default face (Figure 2) is displayed in the *Edit* window; controls and options are displayed in the *Controls* window (Figure 8). The ToonFace menu item has options for selecting and creating new windows and for saving faces. The Edit window contains the face being designed. The Control window, shown in Figure 8, is divided into four regions. At the top (Item Add/Remove) are buttons for adding new polygons (New Attached Polygon, New Free Poly, New Pt. Att. Polygon), and deleting free polygons (Del. Free Poly). Free polygons can also be duplicated (Copy Free Poly). Below these buttons are radio buttons that determine whether points on a polygon are being added, deleted or dragged (Add Points, Delete Points, Drag Points). The options in the *Edit Mode* area determine what kind of polygons we are working with (Hole Polys, Pt. Att. Poly, Sketch), whether we are working with a fully rendered version of the face and choosing colors for the polygons (Colors), whether we are moving the face's control points around (Ctrl. Pts.), or editing the motion ranges of the control points (Ctrl. Pt. Ranges). In addition, we can use the up- and down-arrow keys on the keyboard to select free polygons to edit (Sel/Drag Poly) and swap the order in which they are drawn to make the overlap correctly (Swap Poly Order), also by using the up- and down-arrow keys on the keyboard. In some modes help instructions are printed to the Listener window when they are selected.

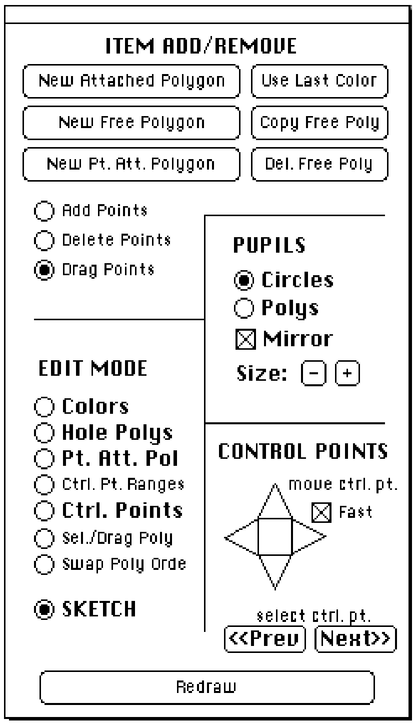

Figure 8. Control panel for the ToonFace Editor (see text). (Pt. Att. Pol = point attached polygons, Ctrl. Pt. = control points; Sel. = select).

The Pupil area allows a user to select polygons or circles as the pupils, as well as increase and decrease their size. With the "mirror" option checked, everything done with one pupil applies to the other as well. The "Control Point" area allows a user to select a control point (select ctrl. pt.) and move it around with the arrows (move ctrl. pt.). With the "Fast" option checked, the selected control point moves 5x the normal distance each time the arrows are clicked. In the "Color" mode, colors are chosen

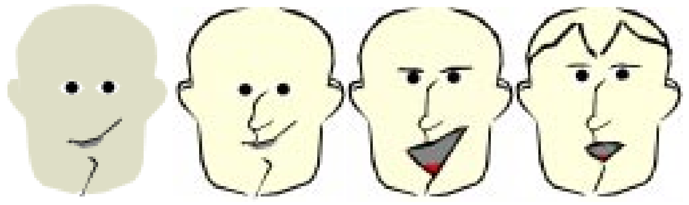

Figure 12. Successive versions of a face as lips (feature-attached polygons), nose and ears (free polygons), tongue (point-attached polygon), and eyebrows (feature-attached polygons) are added.

through the standard Macintosh color pop-up menu by clicking on a polygon; last selected color can be accessed by pressing the "Use Last Color" button before clicking on the desired polygon.

As mentioned before, the Editor allows a user to save a face in two file formats. One is used for the editor program itself; this is simply the format used when an incomplete design is to be stored and read in at a later time for continued editing. It contains Lisp code that can be evaluated directly in the Editor's environmment. These files are designated with the ending ".face". The other is designated ".tfo" *(ToonFace Object)* and can only be read by the Animator.

4.2 The Animator

Once a face has been constructed and saved as a .tfo file, it can be read in by the Animator, which runs on an SGI Indigo2. The Animator is run by calling "TFAnimator *name.tfo"* from the command line, where *name.tfo* is the name of a ToonFace Object file. It then asks for a socket number to use, and once this has been given (e.g. 4050), it waits for a client connection via a standard TCP/IP socket.

The Animator has a defined interface for receiving commands from control programs over the network connection. Commands take the form

```
CODE Controlpoint direction abs-pos exec-time
```

where Controlpoint is the address for a particular motor (a number from 0 to 21), direction is horizontal (= 0) or vertical (= 0), exec-time is the amount of time, in milliseconds, the motion should take, and abs-pos is the absolute position that the motor should have moved the control point to when done. CODE is simply a code that tells the Animator where a record starts (CODE = # for motors and $ for speech). All values are sent over the net in ASCII. Since typically no more than 5-10 records are sent to the Animator per second, the extra number of bits involved in sending ASCII (beyond a defined bitstream) is acceptable.

Head turning and nodding is done through control point 17. Just like other control points, the position of the face when turning is absolute, but here the numbers are given in degrees from center (looking straight out of the screen): positive to the left and negative to the right; negative down and positive up.

The Animator also allows a face to speak, via a DecTalk™ speech synthesizer connected to a serial-port. Speech is coded with a $ at the beginning of the record and replaces abs-pos with a string to be sent to the synthesizer. The Animator will count the syllables in the string and roughly synchronize the mouth movements with its auditory presentation.

The ToonFace Animator can be tested manually by doing

```
>telnet "name-of-machine-running-ToonFace" 4050
```

```
        >
```

at the command line after starting up the program, thus manually connecting to the program, and then giving the command

```
        ># 3 1 100 500
        >
```

which should move control point Brl (3), vertically (1), to its maximum position (100) in 500 milliseconds. The following command will make it speak:

```
        >$ "Hello world"
        >
```

5. THE ToonFace CODING SCHEME: A COMPARISON TO FACS

The Facial Action Coding System (FACS) [Ekman & Friesen 1978] is a system designed for empirical coding of human facial expressions. The FACS model is based on a simplification of the muscle actions involved in producing human facial expression, where muscles are grouped together into what the authors call *Action Units*. Waite [1989] modeled a human face based on a control structure that incorporates several of the action units described in Ekman & Friesen [1978]. In her system, the action units are represented by collections of data points which are covered by a single rendered surface that mimics human skin. The approach taken does not automatically solve how to draw the eyes, control gaze, or add other decorative features (such as ears or hair) to the rendered face. Because the system relies on a model of muscles and bone structure, it is computationally intensive. More recently, Takeuchi & Nagao [1993] describe a system that tries to model a real face in three dimensions based on a similar approach, and Essa [1995, Essa et al. 1994] describes a computational extension to FACS.

The ToonFace coding scheme is not intended to be a competitor to FACS—it simply provides a new way to code facial expressions that requires less detail. Control points were selected to maximize the expressivity/complexity tradeoff. Compared to prior computer systems based on FACS, ToonFace allows for animation with more of a cartoon style look. The motivation for the ToonFace control scheme has already been discussed. However, a comparison to FACS may help the interested reader get a better understanding of the limits and possibilities of this scheme. It should be noted that since the FACS coding scheme is quite complex, the FACS Manual [Ekman & Friesen 1978] is recommended for those who wish to seek a thorough understanding of the issue.

ToonFace is a considerable simplification of FACS, but it is precisely for this reason that it is an attractive alternative. The head motions of humans have three degrees of freedom: head turn, medial (forward-backward) head tilt , lateral (side to side) head tilt. ToonFace simplifies this into two degrees of freedom, eliminating the lateral head tilt. For the upper face, the only features that are identical between the two are the eyes, which have 2 df each. Action unit (AU) 1 (inner brow raiser) and AU 4 (inner brow lowerer) are represented in ToonFace by Bm, with AU 4 approximated by motor Bm having an extended range downward (this depends on the particular face design). AU 2 (outer brow raiser) is approximated by motors Bc and Bl, which also help in capturing motions involving AU 1. Eu, or Eu and El together, approximates the following AUs: AU 5 (upper lid raiser), AU 7 (lid tightener), AU 41 (lid droop), AU 42 (eye slit), AU 43 (eyes closed), AU 44 (squint), AU 45 (blink), and AU 46 (wink). The only one left out from the upper face is AU 6, cheek raiser and lid compressor.

For the lower face, AUs 9 (nose wrinkler), 10 (upper lip raiser) and 17 (chin raiser) are not addressed in ToonFace. Ml represents the motions involving AUs 15 (vertical lip corner depressor), 25 (vertical lips part) and 26 (jaw drop). No differentiation is made between AU 26 and AU 27 (vertical mouth stretch), since the jaw is not modeled separately from the lower lip. Ml and Mr together can approximate the AUs 20 (horizontal lip stretcher) and 14 (dimpler), as well as what Ekman and Friesen [1978] call "oblique" actions—pulling out and up diagonally on the corners of the mouth.

Of course the ToonFace scheme provides nowhere near an exact match to the action of a human face (for which even FACS is a simplification), but that is a problem all computer graphics schemes to date have in common, to various degrees. Where the ToonFace scheme falls especially short is in facial expression involving the physics of skin contraction and excessive exertion of muscle force, and in the combinatorial explosion possible with combinations of the numerous action units included in FACS. With patience, a skilled ToonFace designer could possibly approximate FACS better than indicated here, but that would be going against its design philosophy, which is simply to get a handful of usable facial expressions relevant to multimodal dialogue, while allowing for a playful design that doesn't get the user's expactations up.

6. Applications and Future Enhancements

ToonFace is currently being used as a component in a system the author is building, called Ymir (pronounced *e-mir)*, which is intended to control the behavior of semi-autonomous computer-enacted characters in real-time interaction with people. This system employs a library of motor actions that can be triggered from other system components. The system is optimized for real-time interaction and can thus be used in a number of situations requiring fast responses. Other planned uses of ToonFace include real-time avatar control over web-based applications and storytelling agents.

The ToonFace system is primarily a research tool. As such, it is still missing a number of features that would be desirable and not too difficult to implement. For the Editor, a useful feature would for example be multiple-level UNDOs, as well as improved user interface layout. Also, adding animation libraries to the Editor would help a designer envision what a face looks like when it moves. Currently the animator has no user interface for adjusting such things as background color, size of the face, or window. These would all make the system easier to use. Looking further along, control points allowing the nose and ears to move would extend the kinds of creatures that can be designed in the system. A feature that allowed a face to be texture-mapped onto three-dimensional shapes would of course improve the look of the system quite a bit. The control point scheme described here is easily applicable to more conventional three-dimensional computer graphics, keeping the simplicity without compromizing facial expression.

Lastly, an interesting—and useful—addition would be a mechanism to adjust the face's direction of gaze as it appears to the viewer; research has shown that factors such as face curvature, pupil placement and screen curvature interact in determining where a two-dimensional projection of a face seems to be looking, from the observer's point of view [Anstis et al. 1969]. The same would apply to head motion. This is especially important for systems that track a user's line of gaze [Bers 1995a, 1995b, Koons & Thórisson 1993] and thus allow for reciprocal behavior from the machine.

ACKNOWLEDGMENTS

The author would like to thank Justine Cassell for making this report possible. The work described has also benefited from interactions with Richard A. Bolt and Pattie Maes. Steven Levis assisted with the implementation of the ToonFace Editor and the polygon interpolation methods; Calvin Yuen, Roland Paul, Nimrod Warshawsky and Hannes Högni Vilhjálmsson helped with implementing, debugging and improving the ToonFace Animator, making it a usable tool. Hannes Högni Vilhjálmsson, Scott Provost, Marina Umaschi and Justine Cassell offered valuable comments on this report.

This research was funded by the Advanced Research Projects Agency (ARPA) under Rome Laboratories, contracts F30602-89-C-0022 and FC30602-92-C-0141, by Thomson-CSF and by Media Laboratory discretionary funds.

REFERENCES

Anstis, S. M., Mayhew, J. W. & Morley, T. (1969). The Perception of Where a Face or Television ‘Portrait’ is Looking. *American Journal of Psychology*, 82, 474-89.

Bers, J. (1995a). A Geometric Model of a User’s Upper Body. M.I.T. Media Laboratory, Advanced Human Interface Group Technical Report 1-95.

Bers, J. (1995b). Direction Animated Creatures through Gesture and Speech. Master’s Thesis, Media Arts and Sciences, Massachusetts Institute of Technology, Media Laboratory.

Bolt, R. A. & Herranz, E. (1992). Giving Directions to Computers via Speech, Gesture and Gaze. *Proceedings of UIST* ‘92.

Bolt, R. A. (1980). "Put-That-There": Voice and Gesture at the Graphics Interface. *Computer Graphics*, 14(3), 262-70.

Brennan, S. (1985). The Caricature Generator. *Leonardo*, 18, 170-178.

Britton, B. C. J. (1991). Enhancing Computer-Human Interaction With Animated Facial Expressions. Master's Thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Chin, D. N. (1991). Intelligent Interfaces as Agents. In J. W. Sullivan & S. W. Tyler (eds.), *Intelligent User Interfaces*, 177-206. New York, NY: Addison-Wesley Publishing Company.

Crowston, K. & Malone, T. W. (1988). Intelligent Software Agents. *Byte*, Dec., 267-271.

Ekman, P. & Friesen, W. (1978). *Facial Action Coding System*. Palo Alto, CA: Consulting Psychologists Press.

Essa, I. A. (1995). Analysis, Interpretation and Synthesis of Facial Expressions. Ph.D. Thesis, Media Arts and Sciences, Massachusetts Institute of Technology, February. M.I.T. Media Laboratory, Perceptual Computing Technical Report #303.

Essa, I., Darrell, T. & Pentland, A. (1994). Modeling and Interactive Animaiton of Facial Expression using Vision. *M.I.T. Media Laboratory Perceptual Computing Section Technical Report No. 256*.

Koons, D. B. & Thórisson, K. R. (1993). Estimating Direction of Gaze in Multi-Modal Context. Paper presented at *3CYBERCONF—The Third International Conference on Cyberspace*, Austin, Texas, May 13-14.

Koons, D. B., Sparrell, C. J. & Thórisson, K. R. (1993). Integrating Simultaneous Input from Speech, Gaze and Hand Gestures. In M. T. Maybury (Ed.), *Intelligent Multi-Media Interfaces,* 252-276. AAAI/MIT Press.

Laurel, B. (1990). Interface agents: Metaphors with character. In B. Laurel (ed.) *The Art of Human-Computer Interface Design*, 355-365. Reading, MA: Addison-Wesley Publishing Co.

Librande, S. (1992). Example-Based Character Drawing. Master’s Thesis, Massachusetts Institute of Technology. Cambridge, MA.

Macintosh Common Lisp Reference. Apple Computer, 1990.

Mochizuki, K., Takemura, H. & Kishino, F. (1992). Object Manipulation and Layout in a 3-D Virtual Space Using a Combination of Natural Language and Hand Pointing. *SPIE*, Vol. 1828, *Sensor Fusion V*, 106-113.

Nagao, K. & Takeuchi, A. (1994). Social Interaction: Multimodal Conversation with Social Agents. Sony Computer Science Laboratory technical report, SCSL-TR-94-005.

Neal, J. G., & Shapiro, S. C. (1991). Intelligent Multi-Media Interface Technology. In J. W. Sullivan & S. W. Tyler (eds.), *Intelligent User Interfaces*, 11-43. New York: ACM Press, Addison-Wesley Publishing Company.

Neider, J., Davis, T. & Woo, M. (1993). *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Relase 1*. Reading, MA: Addison-Wesley Publishing Co.

Steele, G. L. Jr. (1990). *Common Lisp the Language*, Second Edition. Cambridge, Massachusetts: Digital Equipment Corporation.

Takeuchi, A. & Nagao, K. (1993). Communicative Facial Displays as a New Conversational Modality. *Proceedings of InterCHI,* Amsterdam, April, 187-193.

Terzopoulos, D. & Waters, K. (1993). Analysis and Synthesis of Facial Image Sequences Using Physical and Anatomical Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 15 (6), 569-579 .

Thórisson, K. R. (1993). Dialogue Control in Social Interface Agents. *InterCHI Adjunct Proceedings*, Amsterdam, April, 139-140.

Thórisson, K. R. (1994a). Face-to-Face Communication with Computer Agents. *AAAI Spring Symposium on Believable Agents Working Notes*, Stanford University, California, March 19-20, 86-90.

Thórisson, K. R. (1994b). Computational Characteristics of Multimodal Dialogue. *AAAI Fall Symposium on Embodied Language and Action Working Notes*, Massachusetts Institute of Technology, November, 102-108.

Thórisson, K. R., Koons, D. B. & Bolt, R. A. (1992). Multi-Modal Natural Dialogue. *SIGCHI Proceedings '92*, April, 653-4. New York: ACM Press.

Tyler, S. W., Schlossberg, J. L., & Cook, L. K. (1991). CHORIS: An Intelligent Interface Architecture for Multimodal Interaction. In *AAAI '91 Workshop Notes*, 99-106.

Waite, C. T. (1989). The facial action control editor, FACE: A parametric facial expression editor for computer generated animation. Master's Thesis, Media Arts and Sciences, Massachusetts Institute of Technology.

Waters, K. & Terzopoulos, D. (1991). Modelling and Animating Faces using Scanned Data. *Journal of Visualization and Computer Animation*, vol. 2, 123-128.

Waters, K. (1987). A Muscle Model for Animating Three-Dimensional Facial Expression. *Computer Graphics*, vol. 21 (4), 17-24.