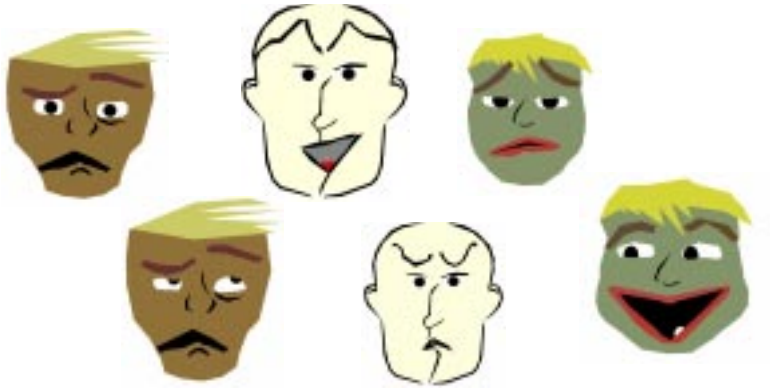
Character Animation

APPENDIX A1

This appendix describes ToonFace, the system used for animating Gandalf's face and hand. It employs a simple scheme for generating effective facial animation (Figure A1-1). It differs from prior efforts for character animation primarily in its simplicity and its way of representing facial features. The next section discusses the background for this work, as well as its motivation and goals. Section A1.2 describes the particulars of the drawing and animation routines. Section A1.3 gives a comparison between ToonFace and the Facial Action Coding Scheme (FACS, Ekman & Friesen 1978). Lastly, current applications and future enhancements are described in section A1.4. A quick user guide to the Editor and Animator are found in Thórisson [1996].

FIGURE A1-1. Examples of faces and expressions generated in ToonFace.



A1.1 Background, Motivation, Goals

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. As the modes of speech, gesture and gaze become a routine part of the computer interface [Thórisson 1995, Koons et al. 1993, Bolt & Herranz 1992, Bolt 1980, 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, Waite 1989] and computer-manipulated images of real faces [NASA Tech Briefs 1995, Takeuchi & Nagao 1993] is that their expressions is often strange looking or 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, 1994, 1994a, 1993a, 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.

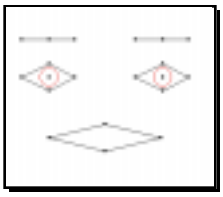


FIGURE A1-2. In ToonFace, seven objects comprise the animated parts of the face: Two eye brows, eyes and pupils, and one mouth. Control points (shown as dots) can be positioned anywhere within the face, by selecting and moving them with the mouse.

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 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.

A1.2 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.

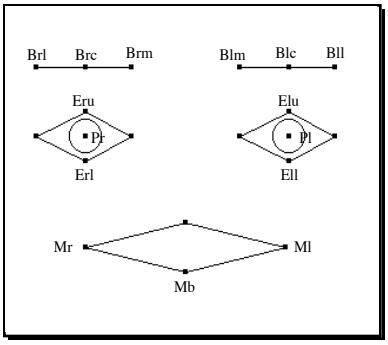


FIGURE A1-3. Codes used for the animated control points (seealso Figure 8-11 on page 123).

A1.2.1 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).

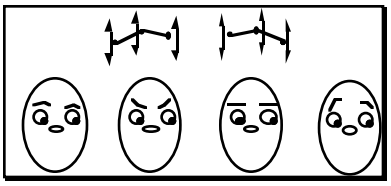


FIGURE A1-4. Eye brows have three control points, each with one degree of freedom in the vertical.

Control points that can be animated are given the codes shown in Figure 8-11 on page 123. These points were selected to maximize the expressive/complexity tradeoff. In the case of points that can move in two dimensions, each dimension is denoted as either "h" for horizontal or "v" for vertical. 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]

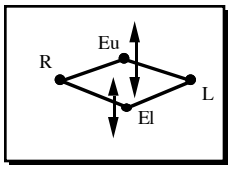
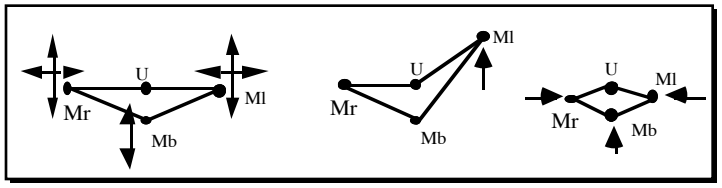


FIGURE A1-6. Each eye has four control points, but only two of those move. Upper (Eu) and lower (El) control points have one degree of freedom each in the vertical.

FIGURE A1-5. 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.



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]

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 A1-1 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 on and dragging the ends of these lines.

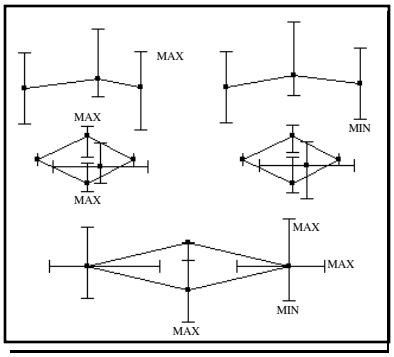


FIGURE A1-7. Limits of control point movement and direction of their dimensions.

A1.2.2 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, 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.

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 A1-8.

FIGURE A1-8. Free polygons are used for objects that don't have to move relative to others, like hats, hair, nose and ears.

Feature-attached Polygons

These polygons are associated with a whole feature. An example is a polygon representing an eye brow (Figure A1-9). These polygons are animated in relation to the whole feature: if one point in the feature moves, all the points on that polygon are recalculated and redrawn: as a result, the polygon changes shape.

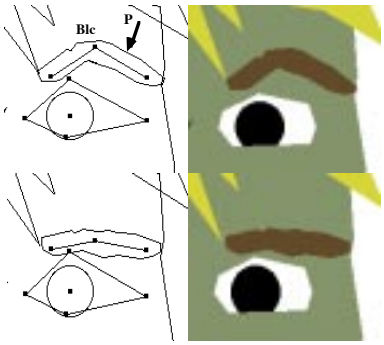
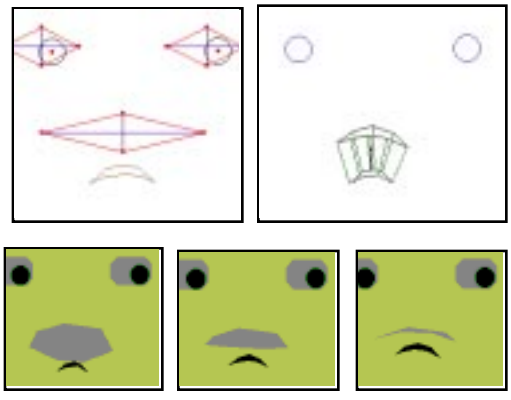


FIGURE A1-9. As the central control point on the left eye brow (Blc) is moved down, the vertices of its attached polygon (P) are recalculated according to how the angle of the lines between the control points changes. The left side shows the control points of the eye brow with connecting lines, the right side shows the polygons when filled.

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 A1-9). When the control point is moved during animation, a linear interpolation is performed between the polygon's two states.

FIGURE A1-10. 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 predefined 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)

A1.2.3 Interpolation Algorithms

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

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:

$$SL = (V1 - V0) / (H1 - H0) \quad \textbf{(A1.1)}$$

The y-intercept of line L1 is given by:

$$IYL = V0 - (SL * H0) \quad \textbf{(A1.2)}$$

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

P = {V1, V2, V3, ... }

V = {X , Y}

$$X = H0 + (VRL * (H1 - H0)) \quad \textbf{(A1.3)}$$

$$Y = (X * SL) + IYL + D \quad \textbf{(A1.4)}$$

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 = {MIN-X , MIN-Y, MAX-X, MAX-Y}

$$X = V_{MIN}\text{-}X + (P_{CTRL} * (V_{MAX}\text{-}X - V_{MIN}\text{-}X)) \quad \textbf{(A1.5)}$$

$$Y = V_{MIN}\text{-}Y + (P_{CTRL} * (V_{MAX}\text{-}Y - V_{MIN}\text{-}Y)) \quad \textbf{(A1.6)}$$

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

A1.2.4 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.

A1.3 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.

A1.4 Future Enhancements

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 and thus allow for reciprocal behavior from the machine.