Computer Animation of Human Figures in Conversation and Action

by

john peter lewis

B.S., Johns Hopkins University
1980

Submitted to the Department of Architecture
in partial fulfillment of
the requirements for
the degree of

Master of Science in Visual Studies

at the

Massachusetts Institute of Technology

June 8, 1984

Signature of author . . . . . .
John Lewis
Department of Architecture
Friday, 8 June 1984

Certified by . . . . . .
Professor Patrick Purcell
Visiting Associate Professor of Computer Graphics
Thesis Supervisor

Accepted by . . . . . .
Professor Nicholas Negroponte
Chairman, Departmental Committee for Graduate Students

Computer Animation of Human Figures in Conversation and Action

by
john peter lewis

Submitted to the Department of Architecture on June 6, 1984
in partial fulfillment of the requirements for the degree of
Master of Science in Visual Studies.

Abstract

Viable articulated computer-graphic representations of the human figure have recently been developed by O'Rourke, Zeltzer, and others. In this work, a figure implemented by Maxwell provides the starting point for the development of tools for controlling the movement and action of figures in a simulated three-dimensional environment.

The figure's representational quality is improved for the purpose of animation, and its capabilities are extended to allow multiple figures to follow arbitrary paths, with posture and movement determined by any combination of key-frames, body-tracking, and algorithmic movement description. Objects in the figure's visual environment are designed using a program for computer graphic sculpture. A sophisticated computer sound synthesis system was implemented and provided the basis for a script-driven multiprocess approach to specifying the interactions of multiple figures in a changing environment.

The resulting system, incorporating figures in an animated visual environment with coordinated sound, may be considered as a vehicle for realizing "electronic cinema". While the animation scripts essentially define a specialized non-procedural programming language, knowledge of a general (procedural) computer language is not required, and figure animations have been realized by artists and filmmakers having no previous background in three-dimensional computer graphics.

Thesis Supervisor: Patrick Purcell
Title: Visiting Associate Professor of Computer Graphics

The work reported herein was supported by the NHK Broadcasting Corporation of Japan

Table of Contents

Introduction

"Computer graphics has not yet delivered fully convincing animations of the human figure." --Norman Badler [Badler 84].

1 Computer Graphic Models of the Human Figure

The computer modelling of the human figure has been undertaken for a variety of purposes, including movement analysis (kinesiology), biomechanics, figure animation, the development of movement notations, and ergonomic research. While the visual representation and movement capabilities of these models varies considerably with their purpose,

1.1 Boeing Man and SAMMIE

Early computer graphic body models such as the Boeing Man and SAMMIE were developed for ergonomic evaluations [Fetter 82, Kingsley 81]. These models emphasized correct proportions and volumes, and usually incorporated the anatomical limitations of movement in the form of joint angle and self-intersection constraints. The figure was positioned by specifying the rotation angles at each joint, and movements were generated using the specified postures as key frames. The figures were visually represented by wireframes delimiting the body volumes. The Boeing Man (Figure 1) has been used in applications such as aircraft cockpit design and crash simulations.

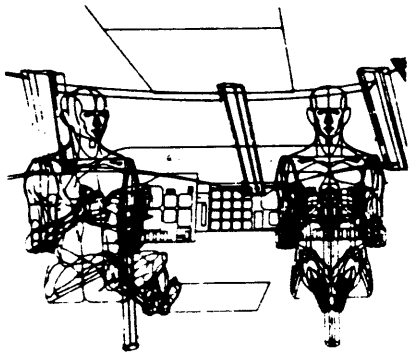

Figure 1: The Boeing Man as pilot

1.2 Bubble Man

O'Rourke and Badler developed the versatile Bubble Man as the primary figure model for the human modelling project at the University of Pennsylvania [Badler 79a, Badler 80]. The Bubble Man's body is represented by spheres of varying radii arranged on a skeletal stick figure. This representation partially conveys the volume and contours of the figure.

The use of the sphere as the modelling primitive has the advantages that body self-intersection tests are reduced to a sequence of sphere intersection tests, and that the perspective projection of a sphere from any viewpoint may usefully be considered to be a circle (in fact, the linear perspective of a sphere oblique to the view direction is an oval-- Raphael noted and corrected this 'anomaly' in portraying the spherical column heads in the *School of Athens*). The sphere is also simple to render and the Bubble Man has lent itself to portrayal of cast shadows and real-time animation [Badler 79a], and microcomputer implementation [Calvert 82].

The stick figure which provides the 'skeleton' of the Bubble Man is implemented with what might be termed an *articulation tree.* This is a simple tree data structure whose nodes are 4x4 transformation matrices. Each node corresponds roughly to a joint and its distal limb segment (if any). The transformation matrix at each node specifies the rotation of the distal limb segment relative to the proximal limb segment, and the length of the distal limb. The limb segments in this description are those of the stick figure rather than the human body, so the hips (for example) are considered "limbs" (Figure 2). The position of any joint is determined by walking the tree while concatenating the translation matrices at each node, and concatenating the resulting matrix with the world (position and orientation) matrix. [Badler 79a, O'Rourke 80a] present a detailed description and Pascal code for the tree-walking procedure.

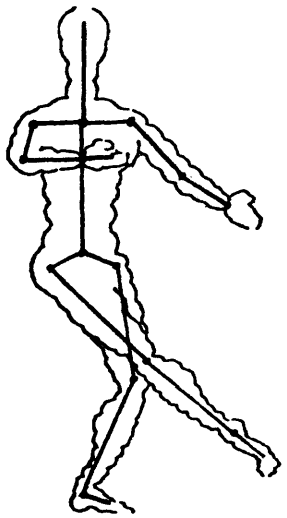

Figure 2: A line drawing version of the Bubble Man, showing the stick figure "skeleton"

The *articulation tree* has several advantages.

- Limbs are treated as units rather than as separate entities, since changing the rotation angles at a proximal node will rotate the limb consisting of the sons of that node in the tree.

- The articulation tree is reminiscent of polar coordinates in its separation of (limb) length and (joint) angle. The posture and movement of the stick figure are specified in terms of joint angles, so a particular posture or movement, once designed, can be applied to stick figures of different proportions (this would not be true of a figure having joint locations specified as cartesian coordinates).

- The consistent tree structure simplifies implementation.

1.3 The AMG Cloud Figure

The goal of the *graphical marionette* project at the MIT Architecture Machine Group is an "electronic puppet" figure which can be manipulated with a body tracker in real time [Bolt 81]. A version of the Bubble Man was implemented at the Architecture Machine Group by Maxwell [Maxwell 82] and provided the starting point for the graphical marionette project and the work described in this thesis. The spheroidal modelling of the the body contours is simplified to an ellipsoid describing each limb segment, and the self-intersection tests were removed.

The *cloud figure* representation of this implementation is a major innovation with respect to the Bubble Man and an earlier ellipsoidal figure [Herbison-Evans 78]. The ellipsoids are rendered as translucent volumes ("clouds") rather than as surfaces. This *perceptually vague* representation is appropriate to the anatomical simplicity of the body model (this point is discussed further in chapter two). A translucent representation also eliminates the need for hidden surface removal.

While it is easy to determine whether a point in three- or two-dimensional space lies within an ellipsoid or its perspective projection, it is more difficult to provide a raster-scan algorithm for rendering the projection of the ellipsoid. Maxwell's particle system ellipsoids are somewhat of a Monte Carlo approach to this problem. A statistical sampling of points within the ellipsoid are projected to form the percept of a translucent cloud. The particle system cloud is easy to implement, and the apparent volume density within the cloud (and consequently its shape) can be controlled as a function of particle position within the local (limb) coordinate system.

This algorithm has several drawbacks for animation, however. The apparent density of the figure is reduces as the figure approaches the viewer, and (conversely) the figure becomes solid as it recedes. The cloud particles are very susceptible to aliasing, so the figure acquires a 'sandpaper' texture when it moves. The individual projection (and anti-aliasing) of the particles is quite time consuming. In view of these difficulties, an image-space (raster-scan) Gaussian elipsoid algorithm was implemented to represent the figure for the purpose of animation (see the description of the figure program in Appendix C).

1.4 Zeltzer's Skeleton

While the cloud figure is perceptually and aesthetically satisfying (and probably more so than any of the other existing figure models), viewers have a strong appetite for additional detail and realism. Fetter (one of the originators of Boeing Man) has recently developed a polygonal model of the body [Fetter 82]. A successful articulated solid body model has not yet appeared, however, and probably will not be forthcoming in the near future. The existing computer graphic solid modelling techniques cannot describe a non-rigid, deforming surface such as the human body in motion.

Zeltzer and other workers at the Computer Graphics Research Group at Ohio State have taken a different approach to the problem of visually representing the figure. They chose the human skeleton as a figure representation which may be approached in detail with current solid modelling techniques. Zeltzer has also improved the articulation tree implementation by distinguishing between supported and supporting limbs [Zeltzer 82a]. When the joint angles of a supporting limb are changed, the rest of the body should move with respect to the limb, rather than the limb moving.

2 The Specification of Articulated Motion

A variety of approaches to specifying the motion of the figure have been tried. A flexible figure model, once implemented, may be directed by any of these methods, or a combination of them:

- *Dynamic movement simulation* is perhaps the "deepest" approach, in which the movement is generated starting with a simulation of the force of the muscle and the mass which it moves [Pierrynowski 82, Hatze 81].

- *Key frame interpolation* generates a movement by interpolating between two predefined postures. A difficulty with this approach (as with any approach which is not physically motivated) is that there is more than one way of moving from one posture to another, even after anatomical constraints are considered. An interpolated movement may be anatomically possible but nevertheless appear unnatural.

- *Analytical movement description* mathematically characterizes a known movement. This is a "shallow" description which does not incorporate anatomical or physical considerations. Cutting's 'walking algorithm' describes the motion of fifteen key points on a walking figure as a vector-valued oscillation (line-to-space function) [Cutting 78].

- *Movement notation* is a traditional pictogram method of recording movement. Movement simulation systems for Labanotation [Hutchinson 60] have been developed by several researchers [Calvert 82, Badler 79b]. The interpreter for a movement notation language must identify patterns of motion primitives and invoke the corresponding movement macro. The movement macro is a predefined movement specification determined by any of the means described here. The movement notation language is sufficiently detailed that it is possible to construct an event-directed movement specification directly on this basis. An example of this type of specification would be (as part of a 'walk' macro): "Rotate both legs backwards at the hips. When the rearmost leg leaves the ground, bend its knee and swing that leg forward at twice the speed of the backward rotation..." This approach does not incorporate the physical constraints of movement (either via modelling or indirectly by digitization) and consequently it runs the risk of appearing stilted. Zeltzer has demonstrated a convincing walking motion using this approach, however [Zeltzer 82b].

- *Body tracking* digitizes real figure movement, rather than analyzing or (re)constructing it. A model's body is equipped with lights or other easily-detected devices situated at key points on the body, and a tracking computer calculates the motion of these points in three-dimensional space in real time. In one approach the points are determined by stereoscopic triangulation from a pair of digitizing devices. [Ginsberg 83] describes the development of a prototype system of this type for the Architecture Machine figure modelling project. Another approach is *electrogoniometry*, or direct measurement of joint angles. Electrogoniometry is more intrusive than stereoscopic triangulation, but it has been used successfully at the Simon Fraser University to produce a improvisation sequence from the *Nutcracker* ballet.

- *Image analysis* is an alternative to body tracking. The movement is first captured on film, perhaps stereoscopically, and later analyzed frame-by-frame to deduce the three-dimensional location of key points. These points are usually highlighted on the figure's costume. O'Rourke and others have applied pattern recognition techniques to this task [O'Rourke 80a]; Fetter obtained running and high-jump motions by manually rotoscoping several of Muybridge's classic movement studies [Fetter 82].

3 The Figure's Environment

While the specialized applications and extensive development time of a human figure model usually result in a model of a lone figure moving in a void, the current work is distinguished in its attempt to control the action and interaction of multiple figures in an environment.

Our work started with the flexible but isolated figure model implemented by Maxwell. The model was developed to allow the integration of several types of motion control (body tracking, algorithmic movement description, and key-framing), and the figure was placed in three-dimensional computer graphic environment where it can interact with other figures and with objects in the environment. The figure's actions and the behavior of objects in its environment are specified in a script.

Chapter one describes the development of a computer graphic sculpture program. This program provides a fairly flexible means of realizing non-articulated objects for the figure's environment.

Chapter two introduces a new use of the computer graphic person model. Facial animation, movement simulation, and natural language techniques are united to create an interlocutory metaphor for man-computer interaction.

Chapter three describes a computer sound synthesis system. This system provided the inspiration and basis for the work in chapter four, and it also may be used to produce a sophisticated, movement-synchronized sound environment for the figure.

Chapter four describes a facility for scripting the actions of figures in a virtual three-dimensional environment.

The final chapter describes how these tools have been used in the figure modelling project at the Architecture Machine.

Chapter One

A Face for the Graphical Marionette

Providing the graphical marionette figure with a face is a priority. The importance of the face is intuitively evident, and empirically underscored by the existence of an area of the brain which, when damaged, hinders or prevents the holistic identificaton of upright faces. The computer-graphic representation of human faces is a difficult and essentially unsolved problem, however. While the effort to develope a face for the graphical marionette figure was discontinued because it was not practical within the timescale of this thesis, some of the tools developed in this work were later used in the modelling of the figure's environment. The development of these tools will be described initially in the context of their original purpose of facial modelling.

1.1 Three-dimensional Computer Modelling of the Face

A face model for the graphical marionette figure must display different (perspective) aspects of the face as the figure moves in its projected three-dimensional environment. In addition, it is very desirable that the face be able to show emotion or simulated speech. The first of these requirements implies a three-dimensional model of the head; an alternative two-dimensional approach will be described later in this chapter. The facial display of emotion and speech indicate a dynamic model of the face. Together, these requirements mean that previous research which approaches the face as a static or two-dimensional problem may not be immediately applicable to a face model for the graphical marionette.

The three-dimensional modelling of the face is nearly synonymous with the work of Fred Parke. In the early 1970s Parke developed a polygonal model of the face, with provision for animation of expressions and speech [Parke 72]. Parke also developed a stereoscopic method of digitizing a face to determine the vertices of the polygonal model. Each face was digitized in a number of positions which served as key frames for the animation of facial gestures. The face was animated by linear interpolation of the polygonal vertices between frames.

Although Parke's faces resemble the individuals whose faces were digitized, the computer rendered faces are perceived rather as masks or robot heads, and appear disturbingly mechanical (Figure 1-1).

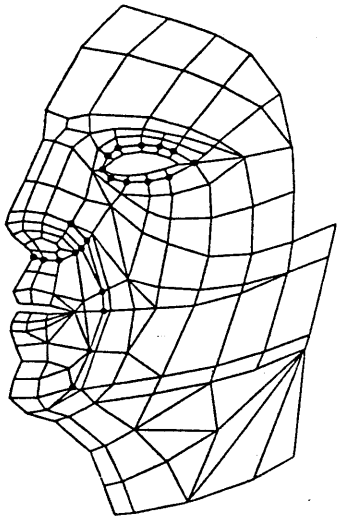

Figure 1-1: Parke's polygonal face

From a technical viewpoint, existing rendering techniques are suited for representing objects having analytically defined, hard surfaces. The current rendering techniques are correspondingly inadequate for representing soft,

irregular, or textured surfaces of natural objects including the human face. Hair is an example of an 'object' which is impossible to render using methods which attempt to represent the object 'surface'.

The surface orientation of current techniques also leads to difficulties when the face is animated. Linear interpolation of the facial surface generally causes the it to pass through unrealistic poses. Platt and Badler recognized this difficulty and approached the facial modelling problem by considering the skin to be an elastic surface moved by the underlying facial muscles; unfortunately this work has not been pursued [Platt 81].

1.2 Gestalt

The mechanical appearance of Parke's faces is best considered as a perceptual problem, however. Compare Figure 1-1 to Figure 1-2.

Figure 1-2: Sketch of a face similar to Figure 1-1

The sketch in Figure 1-2 is incomplete with respect to Figure 1-1 (the

contour lines of Figure 1-1 describe the surface contours of the face, while the lines in Figure 1-2 merely delimit the facial surfaces), yet it is perceptually complete, and more satisfying than Figure 1-1.

The perceptual closure of the sketch is an illustration of gestalt, and the face is a particularly good subject for this illustration. The computer-rendered face exhibits what might be called a 'counter-gestalt': the additional surface information provided in Figure 1-1 is not inaccurate, but its precision and detail incorrectly asserts that this face is exactly as we see it, when in fact this figure is also incomplete. This is a case where the additional information does not help the representation of the face. Figure 1-2 is successful because its "sketchiness" implies that it is not a complete specification of its prototype. Parke describes this phenomenon in his own work [Parke 82]:

> The closer the images get to reality, the more critical the viewer becomes...if the image is clearly supposed to be realistic, the viewer is very sensitive to any flaws.

A similar 'vagueness principle' is responsible for the success of the Architecture Machine's Identidisk [Weil 82]. In this project, the goal is to construct a representation of a remembered or imagined face. Photographs of faces from a 'face library' stored on videodisk are mentally compared with the imagined face. Those faces which were judged to be in some way similar to the desired face are digitized and averaged in the computer to provide a composite face. The photographs in the face library register a normalized facial position to facilitate the averaging.

The Identidisk is thus similar in purpose and effect to the Identikit. The Identikit provides a collection of facial features which are selected and positioned to construct a representation of a remembered face. The firm outlines of the Identikit features imply certainty in the reconstruction. The

composite face generated with the Identidisk, however, is fuzzy as a result of the compositing procedure (intensity averaging), and it consequently operates as an open-ended sketch, informing the viewer that what is seen should not be taken literally.

1.3 Computer-generated Sketch of the Face

Since the development of a realistic face model is certainly beyond the timescale of this thesis, it was decided to use a representation which was intentionally vague. The representation would take the form of a sketch similar to Figure 1-2. This representation avoids the "mechanical certainty" of the polygonal rendering, and it also introduces aesthetic possibilities. Maxwell's "cloud figure" version of the graphical marionette figure is also perceptually vague and in this respect it makes a subtle but important improvement over previous body models [Maxwell 82].

A line drawing is not so vague as to inhibit identification of expressions or character, however. Brennan has explored (among other things) computer-assisted means of portraiture and caricature in the line drawing [Brennan 82]. Our approach to computer-generated sketches is a continuation of Brennan's work using an underlying three-dimensional model.

In the pen-and-ink style of drawing used in Figure 1-2, an object is represented by delineating its projected surface outlines as well as the outlines of the perspective projection of important surface features or colorations. The *surface outline* is fundamental to this representation, and this term will be defined for the purpose of computer implementation as the projection of those points on the object whose tangent is parallel to the local projection ray (a vector from any position to the observer's eye or the center of projection) (Figure 1-3).

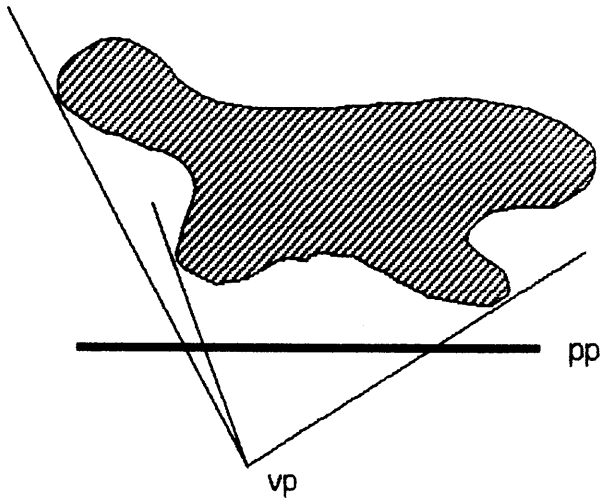

Figure 1-3: Perspective projection of an object:
surface outlines are the projection on the picture plane pp of points on the object which are tangent to the projection ray from the object point to the view point vp.

The sketch is generated from a three-dimensional surface description of the face and head, such as the polygonal representation of Figure 1-1. The three-dimensional surface description is rotated and translated to provide the desired aspect before the sketch is generated. Surface outlines are found by 'crawling' around the surface and collecting points at which the surface is tangent to the projection.

The three-dimensional surface description is composed of vertices, with planar triangular facets implied between the vertices. The points in this surface description which are tangent to a projection ray are the vertices in common to particular pairs of adjacent facets. The criterion for these facet

pairs is that the forward-most facet faces forward with respect to the projection ray, while the rearward facet faces backwards. The facing direction of a facet is determined by the sign of the angle between the surface normal of that facet and the projection ray from any point on that facet (conveniently, one of its three vertices) to the center of projection. This angle may be obtained as the dot product of the surface normal with the projection ray.

The resulting collection of (projected) vertices is arranged in a coherent order and the sketch is obtained by connecting the vertices (Figure 1-4). A number of heuristics were required to implement the tangent determination and subsequent ordering of the projected vertices into outlines.

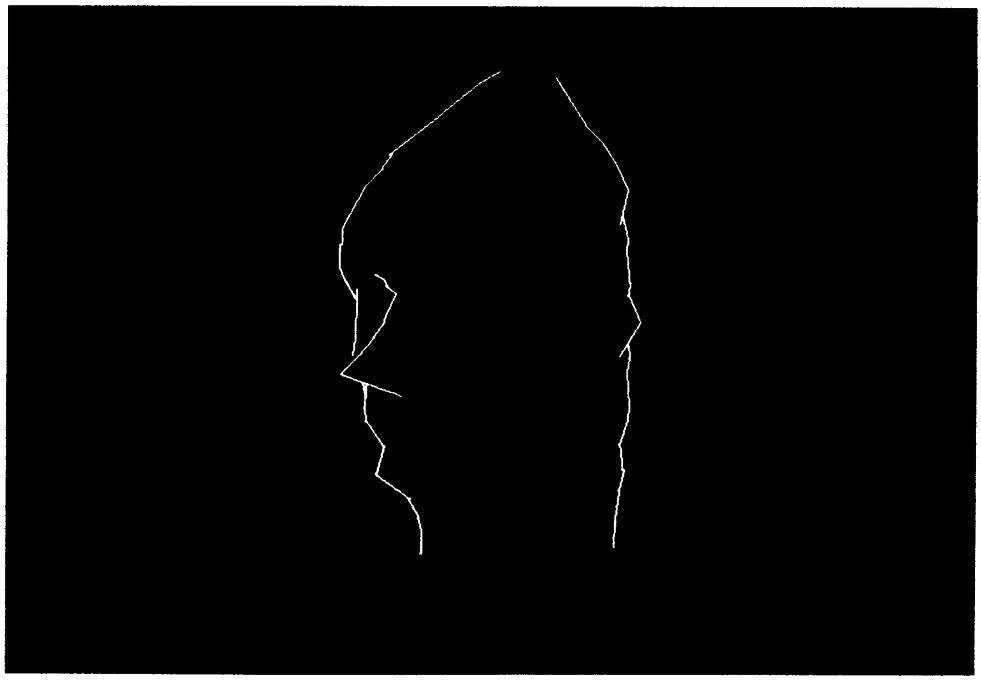

Figure 1-4: Surface outlines identified with the 'tangent-finding crawler' and heuristics.

A procedure for automatically identifying and drawing the surface outlines of objects was implemented (as described above), but this work was abandoned before a complete system for computer-generated sketches could be developed. It was intended to generate surface features and facial expressions for the computer-generated sketches of the face by texture-mapping one-bit line drawings of the desired features onto the three-dimensional surface representation, and projecting these to obtain perspective versions of the features.

The features and expressions were to be developed as drawings (or sequences of drawings) input using a digitizing tablet. Ekman's work provides a pan-cultural taxonomy of facial expressions as well as an underlying "emotion space" which could be used to guide the evolution of facial expressions [Ekman 73, Ekman 75]. The mapping of mouth positions for portraying speech would draw upon the 'lip sync' work at the Architecture Machine Group at MIT [Negroponte 80].

It was also intended that the perceptual vagueness and aesthetic quality of the sketch be emphasized by post-processing the sketch with a computer simulation of human line drawing [Lewis 82], Figure 1-4. This simulation describes the motion of a hand-held pen tracing a path towards a target point, given prescribed initial conditions. The motion is modelled as a feedback system with memory (momentum).

1.4 Surface Reconstruction from Planar Contours

The remainder of this chapter will describe how the three-dimensional surface description of the face was obtained. The modelling tools developed for this purpose are now used to construct objects to populate the graphical marionette figure's environment.

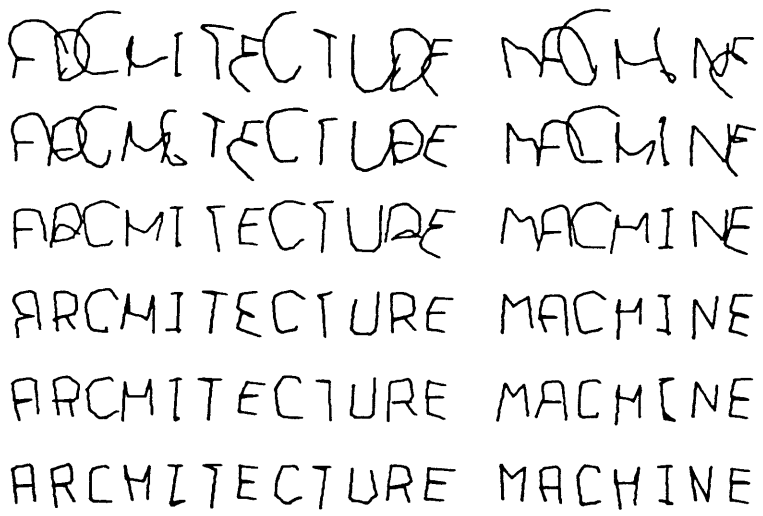

Figure 1-5:CALCOMP plotter text processed with a line drawing simulator. A 'scribble' parameter was incremented to obtain this progression.

The computer modelling of an object such as the face requires techniques different than the analytical tools which are responsible for the spheres, planes, and polyhedra which characterize computer graphics. While it is straightforward to write a program to draw a sphere, the task writing a program to draw a portrait of President Nixon (for example) is nonsensical. A computer model of a face can realistically be obtained by measuring a face and digitizing these measurements. A face is not a particularly easy thing to measure, however.

Parke's stereoscopic method is one successful approach to this problem. In our case, the availability of data describing a head suggested another approach. A company called Robotic Vision Systems uses a automated method to digitize a solid object. A sculpture of the object is then machined [Rongo 82]. A byproduct of this 'solid photograph' is a detailed, digital description of the object surface. The data describing a bust of Professor Nicholas Negroponte of MIT was obtained (Figure 1-6).

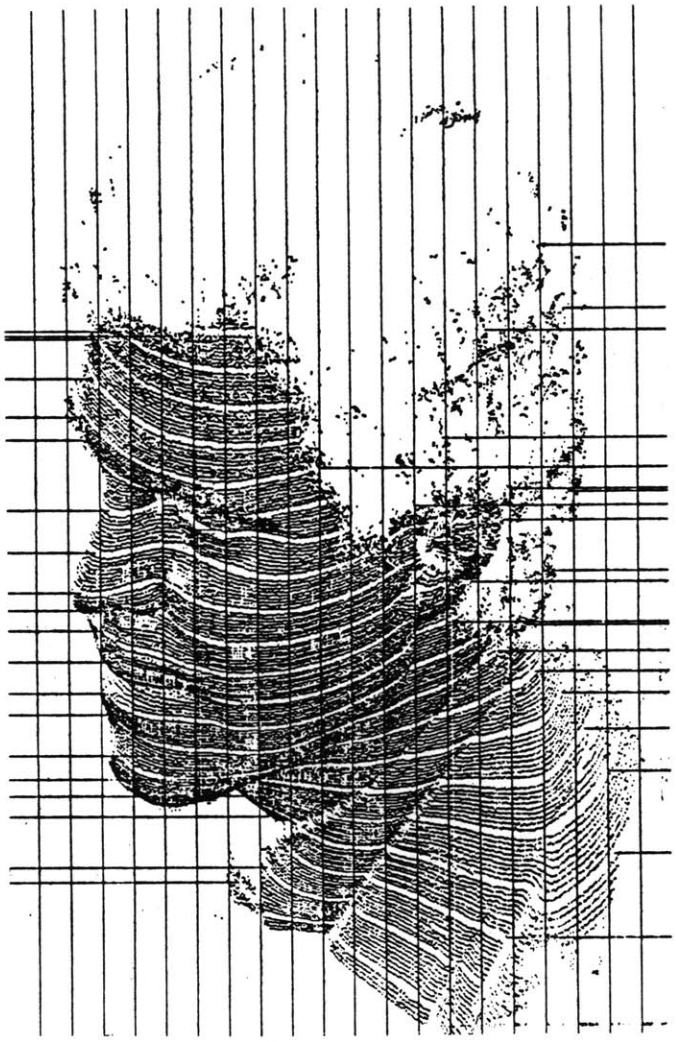

Figure 1-6: Data for a three-dimensional bust of Professor Nicholas Negroponte, from Robotic Vision Systems' solid photography system.

The data takes the form of the coordinates of points located on more than 300 horizontal cross-sections of the head, comprising more than 1,000,000 points in total (the terms section, contour, and slice will be used synonymously hereafter). This amount of data was judged to be impractical in terms of both memory and processing time, so a few thousand points were obtained by roughly sampling the original data.

The points are then connected to form facets defining a surface which approximates the original head. Since there is more than one surface which spans a given set of 3D points (Figure 1-6), this is a heavy computer science problem. Indeed, it has been postulated that the problem of finding the optimal polyhedral approximation of the object described by a set of 3D points is *NP-hard* [O'Rourke 81].

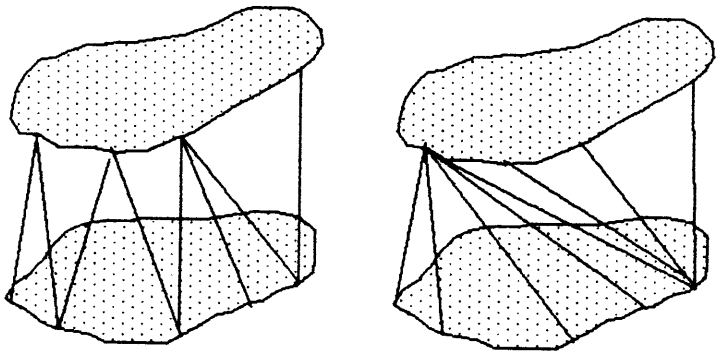

Figure 1-7: The 'tiling problem': a set of points obtained by digitizing sections of a three-dimensional object may be spanned by many different surfaces, some of which approximate the original object

The problem has the following constraints: a point on a particular slice may only be connected to the adjacent points on that slice and to points on the two adjacent slices, i.e., facets are formed between pairs of adjacent slices. Planar facets are desired for convenience in the subsequent (polyhedral) rendering process. Triangular facets are desired since a group of four or more points distributed on two adjacent slices will rarely be contained in a plane.

Additional constraints which define an acceptable (non-self-intersecting) surface are:

- Each *contour segment* (a segment joining two adjacent points from a particular slice) will appear in exactly one facet. The number of facets spanning a pair of adjacent slices will therefore be m+n, (the number of points on the two slices).

- If a *span* (connection of two points) is the left (right) edge of some facet, it will appear as the right (left) edge of exactly one other facet.

With these constraints, the number of acceptable surfaces spanning a pair of adjacent slices having m,n points is

$$A[m,n] = [(m-1) + (n-1)]! \,/\, [(m-1)!\,(n-1)!]$$

A criterion for picking one of these surfaces is desired. Keppel formulated this 'tiling problem' as an equivalent graph theory problem, in which the set of all acceptable surfaces for the pair of slices defined by points $\{P_0,P_1,....,P_{n-1}\}$, $\{Q_0,Q_1,....,Q_{n-1}\}$ are represented by a directed graph G[V,A]. The set of vertices V corresponds to the set of all possible spans between the upper contour and the lower contour. The set of arcs A corresponds to the set of possible facets. An arc is defined as incident from the vertex which represents the left span of the facet to the vertex which represents the right span of the facet. The path of an acceptable surface will have m+n connected arcs (modulo the size of the graph) on a graph of size m,n arcs.

A weight ρ assigned to each arc. The *optimal* tiling is selected from the set of possible tilings using the global weighting function

$$P = \Sigma_{k=1}^{m+n} \rho_k$$

A non-optimal tiling can proceed incrementally by choosing the arc leading from each vertex in sequence as a function of a subset of the undetermined and previously determined arcs. A non-optimal tiling which considers only

the weights attached to the two arcs leading from the current vertex runs in linear $O(m+n)$ time.

The graph-theoretic formulation of the tiling problem is standard, and many solution algorithms and weighting criteria have been proposed [Ganapathy 82, Boissonat 81, O'Rourke 81, Fuchs 77, Keppel 75, Christianson 76]. The global 'brute-force' minimization of surface area (suggested by Keppel) was adopted and implemented. This method requires approximately an hour of CPU time on a 32-bit minicomputer to reconstruct objects such as those in Figure 1-9, containing several thousand facets. Dunbar Birnie implemented several other criteria and solution algorithms. The (local) minimum-edge distance criterion works well and does not require notable amounts of CPU time for the objects which were developed in this work (Figure 1-8).

1.5 Potatoslice: A Program for Computer Graphic Sculpture

The slice reconstruction modelling method and has applications beyond face and head modelling and it is commonly used in medical imaging [Ganapathy 82]. In general, it allows one to design a variety of lumpy, semi-concave objects which do not lend themselves to analytical description.

Once a (polygonal facet) surface has been reconstructed from the slices, it may be rendered using standard methods such as those described in [Foley 82]. Several reconstructed heads are shown in Figure 1-9.

The PotatoSlice program (Appendix A) incorporates several tiling methods with a rendering program. This system may be approached as a tool for sculpture, and artists using it have become skilled at designing fanciful and realistic objects by conceiving the object cross sections. The Cambridge

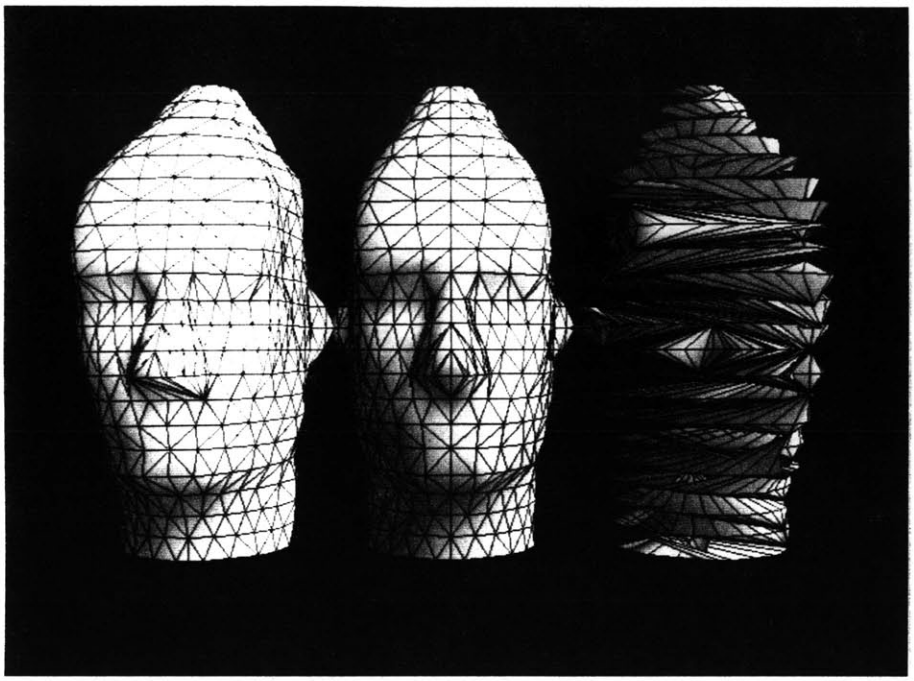

Figure 1-8: Comparison of three criteria for the tiling problem: left, local minimum distance; center, global minimum area; right, local maximum distance (!). Figure by Dunbar Birnie.

sculptor Ralph Helmick uses a physically analogous approach; his large-scale sculptures are produced by laminating plywood slices. The use of this modelling tool to configure the graphical marionette figure's environment will be described in chapter five.

1.6 In Retrospect

While the approach described in this chapter may result in a flexible and perceptually satisfying model of the face, it is far from practical within the time scale of a short Master's thesis. In retrospect, several simplifications may be identified. The surface edge-detection procedure would be easier to implement (though computationally less efficient) by band-pass filtering the

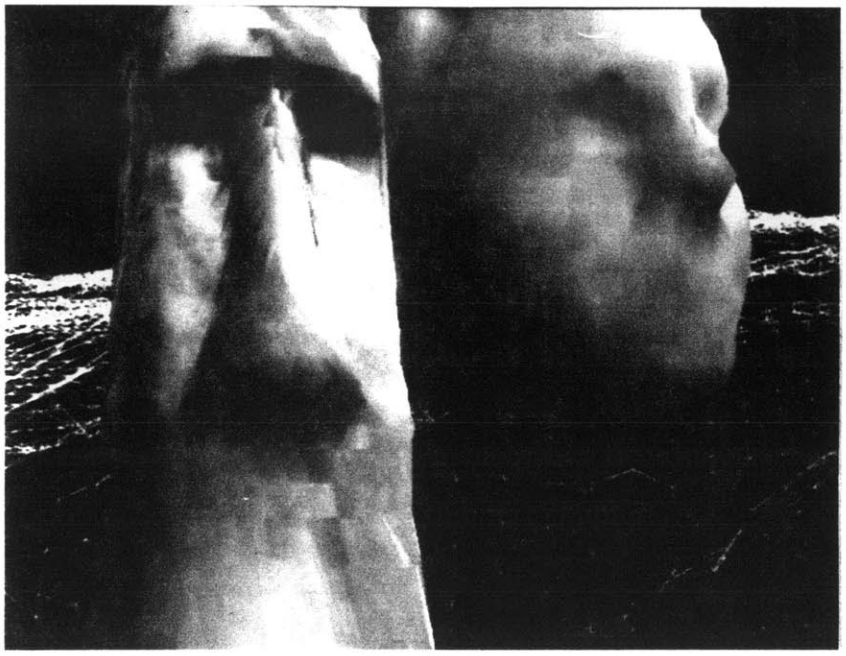

Figure 1-9: Two heads created using the slice reconstruction method described in this chapter. The head on the right was generated using the data displayed in Figure 1-6, while data for the Easter Island head was estimated by the author.

projected, shaded 3D image obtained from standard rendering techniques (Figure 1-9, 1-10).

The requirement that the face be viewable from different directions suggests but does not necessitate a three-dimensional face model. As an approximation, a planar representation of the face could be texture mapped (front projected) onto an ellipsoid 'head'. This technique has been used at the Computer Graphics Lab at the New York Institute of Technology.

A 'graphical robot' which incorporates a real-time two-dimensional facial animation technique will be described in the next chapter.

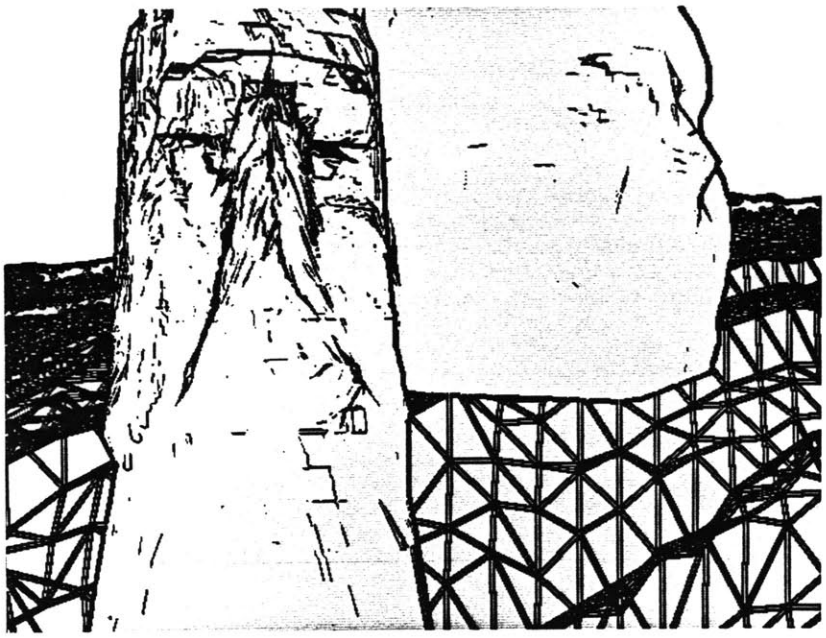

Figure 1-10: A high-pass filter applied to Figure 1-9 emphasizes surface outlines and detail. A band-pass filter (Figure 1-10) produces bolder contours and eliminates less-important surface detail.

This page was intentionally left blank

Chapter Two

Conversations with a Graphical Robot

The graphical robot is an alternative to the graphical marionette, in which the figure is given autonomy and interaction takes the form of purposeful conversation rather than real-time puppetry. In the system described here, purpose and autonomy derive from the robot's role as the interface to conventional (command language driven) programs. A variety of existing tools are combined to create a true conversational interface.

The success of this work is in part indicated by the fact that many observers of the system were initially skeptical of the computer's performance. The key to this success does not reside in improved speech hardware but in software tools which provide a convenient and consistent means of tracking and responding to conversational context. This work was judged to be inappropriate, however, and it was succeeded by the more traditional animation approach described in chapter four.

The following description is taken from [Lewis 84].

ABSTRACT

The man-computer interface has evolved from the teletypewriter to workstations which accomplish a spatial metaphor for data management. In the experimental system described here, speech and graphics techniques are used to produce an interface in the form of a metaphorical person. Interaction takes the form of an unconstrained spoken conversation with a

"graphical robot" whose animated likeness is displayed on a high-resolution computer graphics display. This system is proposed as a prototype of "casual interface" for machines which we do not use often enough to justify learning a command syntax. The realization of such systems assumes the development of limited-vocabulary speaker-independent continuous speech recognizers. The system architecture, performance, and assumptions are discussed.

Introduction

The extent to which metaphors of existing systems have influenced computing is suprising but undeniable. The computer interface modelled on the teletypewriter restricted the interface to a single active line of characters, resulting in line editors, line-oriented languages, and 'linear' (one event at a time) command languages.

Graphic display terminals generalize the potential "interface space" from the line of characters to a planar visual space. Pioneering work at the Xerox Palo Alto Research Center used high resolution bit-mapped displays to create a spatial metaphor in which concurrent processes are seen as spatially separate objects located on a "desktop" display [Kay 77]. The Spatial Data Management System explored navigation through a "dataland" where the data could consist of color images, movies, and sounds, as well as iconic (quality font) representations of alphanumeric data [Bolt 79].

Beyond the developments provoked by spatial metaphors, other uses of graphic displays are slower in evolving. One can speculate, for example, that color, depth representation, and iconography may better represent aspects of program structure and function such as scope, binding, and concurrent process execution.

In this work the power of the metaphor is explicitly acknowledged and a new metaphor is created. The title of this paper derives from the use of the word "soft" to describe computer interfaces whose design considers the computer user. Certainly the softest interface is another person--a programmer who can be instructed to accomplish the desired task.

Soft machine demonstrates an interface to several conventional computer programs (data-base query, electronic mail) which resembles interaction with a person. The interaction is in the form of an unconstrained spoken conversation with an electronic conversational partner (alter ego). The alter ego is an animated person-likeness which "speaks" with a high-quality speech synthesizer and "listens" with a continuous speech recognizer (Figure 2-1). The animated image reflects the activity of the alter ego: the head motion of the likeness is consistent with attentive listening and the lips move with the alter ego's speech. The alter ego's conversational interests and visual character are easily personalized, suggesting 'alter ego' for this conversational partner.

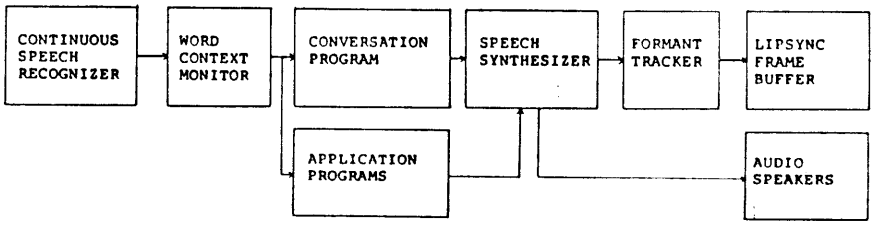

Figure 2-1: Soft Machine system flowchart

2.1 Conversational Technique

The alter ego's conversational ability resides in a pattern-matching script. Patterns consist of ordered lists of words, optionally separated by wildcards (which match anything) or digits (which match a particular number of arbitrary words). Each pattern is associated with a list of responses or response procedures. Patterns are arranged in ascending order of generality. The output of the speech recognizer is tested against the patterns until a match is found. One of the associated responses is selected and either spoken or executed (the lisp EVAL function is used to execute program fragments within the script). As an example, the user might say

"why are you taking so long?"

The corresponding output of the speech recognizer is

(? are you ? ? ?)

which matches the pattern

(1 are you *)

and provokes the response

"Do you think I am?"

The speech recognizer used in this work returns up to five words recognized from continuous speech (one or two wrods per sentence is typical performance) with indication of words which are heard but not recognized.

The conversational script is similar to the Eliza program [Weizenbaum 64] though less sophisticated. The purpose of soft machine's conversational ability is not to simulate conversation with another human but to complete the personal metaphor while functioning as a sophisticated means of conversational prompting and system feedback. Thus, script responses serve in the following roles: 1) To indicate the system's misunderstanding in a

friendly and interesting way. A variety of statements replace the usual "?Unrecognized Command". This role is particularly effect in overcoming the frustration which often results from the limitations of current speech recognition systems. 2) To encourage the user's continued and uninhibited interaction. The user's inquiry or expression of frustration may result in the amusing "clearly this is an existential matter" or the potentially constructive "we were reading your mail" rather than the illegal command syndrome.

Unconstrained (amodal) conversation was a major goal of this work. The user can adress any application program at any point; the currently-active program is interrupted and its state is saved. Amodal conversation is facilitated by the limited vocabulary of speech recognizers. The vocabulary is carefully selected to consist of words which may be used in more than one application, as well as general words which support the conversation. Each word has a context list, and recognized speech is examined by a context monitor program to determine which to program it is addressed. Thus, 'next' may refer to the next piece of mail or the next database entry, or (as a non-sequiteur) become the object of a conversational remark. The context monitor uses a number of obvious heuristics to determine the applicability of the recognized words and to discard words which may be spuriously recognized or are currently inapplicable. Application programs maintain a status message which is spoken if no applicable input is recognized in the course of several conversational exchanges.

A contrived but realistic example of a conversation with Soft Machine follows (recognized words are italicized):

user: *Hello*

machine:

Welcome to the machine.

u: *I* want to see the Rotch database.
m: Speak up.

u: *Show me* the *Rotch* database.
m: The Rotch slideathon is available.

u: buildings in *Massachusetts, 1950* to *1960.*
m: Confirm state Massachusetts.

u: *OK*
m: Confirm dates 1950 to 1960.

u: *OK*
m: Initiating search
Ready.

u: Howmany?
m: What are you talking about?

u: *How* many?
m: There are 646 entries.

u: *Show* them.
m: (first slide appears)

u: *Next* slide.
m: (second slide appears)

u: Read *my* mail.
m: How do you know it is yours?

u: *Shut up.*
m: You do not like me very much.

u: Read *my mail.*
m: You have no unviewed messages.
Last message dated .... (the most recent message is read).

2.2 Real Time Animation of Electronic Persona

The text output by application programs is returned to the alter ego, who reads it to the user. The text is first massaged to translate written conventions into their verbalizations, for example, the electronic mail address "godzilla@mit-pamela" becomes "godzilla at M.I.T. pamela". The text is then output to a high-quality speech synthesizer (Prose 2000).

The speech synthesizer's audio output is simultaneously sent to speakers and analyzed by a real-time formant-tracking computer (Fig. 1). Ten formant configurations are selected, corresponding to visually distinct lip positions. A 1024x1024x8 frame buffer stores up to eighty 128x96x8 images of lip and head movements. Currently two sequences of ten images provide "positive emotion" and "negative emotion" lip positions. The formant tracker reorigins and zooms the appropriate frame buffer images, so that the alter ego appears to speak. This "lipsync" technique was developed at the Architecture Machine Group for limited bandwidth teleconferencing applications [Negroponte 79].

The basic lipsync technique is extended to include limited expression and head movement. When the alter ego is not speaking its head moves in a subtle but animated way characteristic of attentive listening. The remaining frame buffer images (those which are not used for lip positions) include eye and head positions. A transition matrix describes coherent random sequencing of these images.

2.3 Evaluation

Though a system such as this one would make a cumbersome interface to a text editor (for example), it is an attractive interface to a machine which one uses occasionally and does not wish to know in detail. An on-line library catalogue search facility is an example. In one library a catalogue-search terminal was installed adjacent to the card catalogue. Library users were either intimidated by the computer terminal or reluctant to learn its command syntax, and the terminal was removed for indirect access via information-desk personnel (Eisenhower library, Johns Hopkins).

Soft machine is fun to use. Novice users are fascinated by its conversational ability and willingly explore the system. The combination of animation and speech techniques succeed in creating an animated persona.

Suprisingly, a limited vocabulary of several hundred words, if carefully chosen, is quite adequate to support both applications and an interesting conversational capacity. This is because the system does not need to understand all of what is said, but only to recognize words which may be meaningful to application programs, and provide a reasonable conversational response if none are found ("limited recognition"). Appropriate applications have a limited set of commands. Some information retrieval systems are appropriate both in this requirement and in being systems which one might casually encounter. A videodisk-based architectural database was interfaced to Soft Machine. Its input vocabulary of about 100 words comprising several independent keys (state, date in decades, building type) allows access to all of the 5000 entries in the database.

The speech recognizer is nevertheless the weak link in the system. The conversational interface presumes the ability to reliably identify vocabulary

words embedded in speech containing a large percentage of words which are not in the vocabulary. Current continuous speech recognizers perform poorly at this task. In addition, continuous speech recognizers are usually speaker dependent, requiring retraining with each new user. It has been recognized that the parsing of natural language often requires understanding, which in turn may require human-like world knowledge and intelligence. The recent development of finite state (phonemic) probability driven recognition models argues that incremental developments in speech recognition will continue however [Schwartz 84], and a limited-domain speaker-independent recognizer capable of realizing an interface such as Soft Machine may become available long before the problems of language understanding are resolved.

Chapter Three

The Sound Environment

Given the author's interests and skills, it was decided to provide the marionette with a sound environment as well as a visual environment. While the latter is fundamental, the sound environment will be discussed first, in part because this discussion will motivate some of the decisions which affected the design and implementation of the visual environment.

Computer music languages are a good place to start in the development of a computer-generated sound environment. A computer music language can act as a sophisticated studio if it is desired to use natural sound sources. The digital equivalents of sound mixing and splicing are easy to program and have advantages of precision, control, and low noise. Programs to do equalization and effects such as reverberation and chorus are found in some computer music systems.

In addition, a computer music language has facilities to produce novel, imaginary sounds or music. Most computer music systems are oriented toward providing interactive control for novice (computer) users, and those "architectural" features (design and implementation approaches) of the computer music language which enable this control will be adapted to form the basis of the system for configuring and controlling the figure's visual environment. It should be noted that, while computer music languages are intuitive for the non-programmer (and particularly for the person who has experience with analog musical synthesizers), they are essentially primitive programming languages. The common sound-generating modules

(oscillator, noise generator, filter, etc.) are provided as 'pre-programmed subroutines' and the user need only determine the parameters and calling order of these modules. Programming constructs such as iteration and conditional branching are not essential and are not usually provided in the computer music language.

3.1 The Music V Language

The major music languages, including Music 10, Music 11, Music 360, and Cmusic, are based on or developed from the Music V language developed by Max Matthews during the early 1960s [Mathews 69]. Music V divides the sound specification task into an orchestra or "instrument" programs and a score file. The instruments are designed as an ordered list of signal generating modules which receive and update information passed by a subroutine parameter list convention. For example, a very simple (and electronic-sounding) instrument looks like this:[1]

```
begin "i1"
A1      =  oscil(SINE,440,1.0)                    (1)
A2      =  oscil(ENVELOPE,1.0,P4)                 (2)
out     =  out + A1 * A2                          (3)
end
```

Example 1

Statement (1) calls an oscillator to produce a 440 cycles-per-second (cps) "sine" wave with amplitude one. The sine wave is assigned to the variable A1. In statement (2) the waveform "envelope" (defined elsewhere) is oscillated at one cps with the amplitude P4. P4 refers to the fourth parameter of the score statement which will invoke this instrument (this will be

---

[1] The examples in this chapter are written in a hypothetical music language which is essentially similar to Music V, 10, 11, 360, and Cmusic.

described below). In statement (3) the sine wave A1 is modulated by the amplitude envelope A2 and the result is added to OUT which will be sent to the audio speaker.

3.2 Digital Sound Synthesis: Sampling and Quantization

While it is intended that the variables A1,A2,OUT denote continuous signals, in the digital implementation these signals are necessarly generated in a sampled and quantized form. An audio signal is thus represented as a sequence of numbers, which are converted into a voltage (to drive an audio speaker) with a *digital-to-analog converter* (DAC). The fidelity of the resulting sound depends on both the accuracy of the quantization and on the sampling rate. The *Shannon sampling theorem* indicates that frequency components higher than one-half of the sampling rate will not be reproduced correctly [Shannon 49, Stearns 75]. Some individuals can detect frequency components up to about 20,000 cps at a young age, entailing a sampling rate of at least 40,000 cps. Musically interesting frequencies appear to be well below 10,000 cps, however, implying a more economical sampling rate of 20,000 cps. This judgement is best justified by listening to a sound reproduced at various sample rates, but in passing it can be supported by several observations: many people cannot hear above 10,000 cps, the bandwidth of AM radio is about 5000 cps (FM radio has a bandwidth of about 15,000 cps), and the fundamental harmonic of most musical tones is below 1000 cps ("middle C" on the piano is 262 cps).

The commonly available digital-to-analog converters accept signals which are quantized to between eight and sixteen bits. Quantization error most directly affects the dynamic range of the signal, measured as the *signal-to-noise ratio* (SNR). Eight-bit quantization results in a SNR of about 48 decibels, which is

approximately comparable to the quality of telephoned audio signals (a decibel is a logarithm of the ratio of two amplitudes, defined so that +10 decibels is 3.162 = sqrt(10) times the original amplitude [Benade 76]. Sixteen-bit DACS have a SNR of about 96 decibels, which is very high fidelity. Twelve-bit DACS have been used in computer music without causing perceptually obvious quantization effects.

3.3 Specification of Sound Events: the Score File

The instrument in Example 1 is thus called thousands of times per second, and the signal generating modules (OSCIL in the example) are programmed to return a single sample of their respective signals each time they are called. A 'conductor' program reads the score file, determines the starting time and duration of each 'note' in sequence, and calls the appropriate instrument, passing the instrument parameters from the score to the instrument. Example 2 is a sample score:

```
srate=20000;
synth "sine" 1,1.0;
seg "envelope" 0.0,0.0 0.1,1.0 1.0,0.0;
i1 0 1 1000 ;
end;
```

Example 2

This score activates "i1" (the instrument defined in Example 1) at time zero for the period of one (virtual) second (20,000 samples) with the auxiliary parameter P4 = 1000. In this score language, the first token of an instrument statement (beginning with an "i") is the instrument number, and P2 and P3 are conventionally the start time and duration of the event or note defined by the instrument statement. The "synth" statement creates a function table ("sine") and fills it with a Fourier summation having the first partial with amplitude one (a pure sine wave). A finite Fourier summation approximating a square wave would be

```
synth "square" 1,1  3,0.3333,  5,0.2,  7,0.1429;
```

(odd harmonics with 1/N amplitudes). The arguments to the "seg" statement specify the abscissa, ordinate pairs for a line segment (piecewise linear) function. The abscissa and ordinate values are specified in the ranges {0,1} and {-1,1} respectively.

3.4 Function Table Lookup Models

The preceeding discussion implies that signals are stored in function tables and regenerated by table lookup, rather than by direct polynomial expansion approximations. This is done for reasons of speed. In the early 1960s people were conscious of the cost of a computer hour and only the cheapest techniques could be considered. This had a major effect on the development of computer music languages.

The effect of this effect will be considered by considering the traditional implementation of a synthetic guitar approximation (for example) using a Music V-type language. It is common knowledge in musical acoustics that a struck or hammered string generates odd harmonics with 1/N amplitude (a square wave), while a plucked string generates $1/N^2$ odd harmonics (triangle wave). The natural decay of a plucked or struck string is exponential. The sounding box of the guitar acts as a resonator or filter, and this will be approximated with a gentle bandpass filter. This completes our "synthetic guitar".

Unfortunately this approximation does not bear the least resemblance to its prototype. Physical musical instruments are universally complex and nonlinear in their porduction of sound, and the difference between the computer and the physical instrument is as between a toy synthesizer and an

expressive sound. We can attempt to describe the characteristics of musical instruments in various ways; a relatively successful approach is to consider the frequency spectrum of the sound, and its evolution. It is known that for a given (physical) instrument, characteristics of the spectrum will depend on the amplitude, pitch, and performance of each note in a nonlinear way. Thus, the spectrum of a note with the pitch C5 (the note one octave above middle C on the piano) will be distinctly different from the spectrum of a note with pitch C4 which is shifted up one octave in frequency.

It has also been demonstrated that the evolution of the sound spectrum, particularly during the onset of the note, contributes significantly to our perception of timbre and to the identification of the instrument [Stumpf 90]; the spectrum which is constant or which evolves according to a simple formula identifies the instrument as an electronic synthesizer.

In the time domain, it can be observed that physical vibrating systems are not periodic but *almost periodic*. An effect of friction is to slightly alter the harmonic frequencies of a vibrating system, so that they are not in the same phase relationship from one period to the next. The contour of the acoustic wave can change significantly during the course of a note.[2] The effects of the stiffness of the vibrating body and resonant coupling are similar to that of friction in that they remove the musical signal from strict periodicity.

The selection of the oscillator as the fundamental modelling tool and sound source for computer music is suggested by any analysis which considers music to be a fundamentally harmonic or periodic phenomenon. Oscillator

---

[2]The perception of timbre is fairly robust in that the physical correlate of the timbre is usually a rapidly evolving waveform. The perception of timbre might in some cases be equated to the evolution of sounds rather than to a steady-state spectrum.

models are inappropriate, however, for modelling transient or inharmonic sound features which may be perceptually important even if they do not by themselves define a musical sound. For example, the spectrum of many instruments appears as harmonic peaks above a resonant "noise floor" (Figure 3-1). The periodicity and linearity entailed by the function table lookup method as well as the triangle and square wave characterizations of musical acoustics (mentioned previously) are approximations to physical sound production.

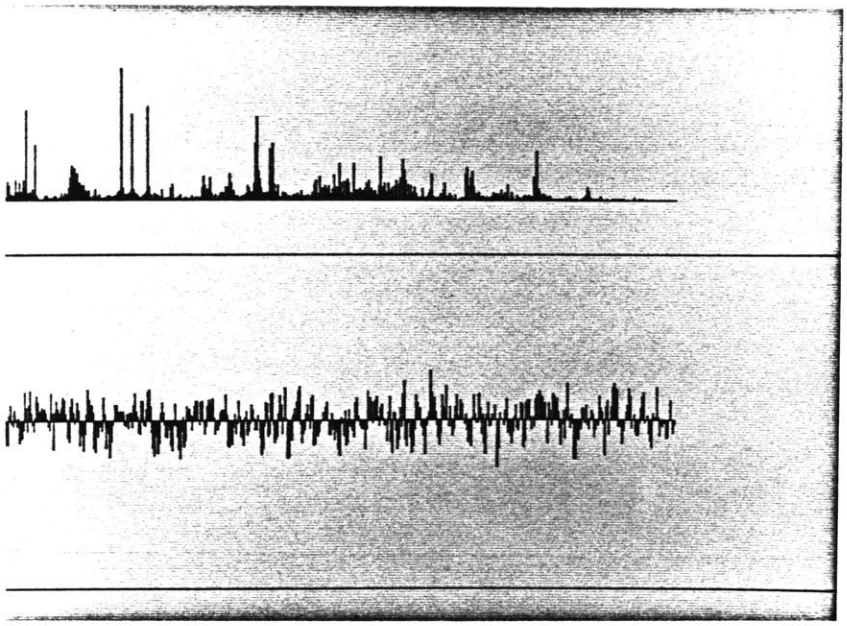

Figure 3-1: The spectrum of real music: a guitar passage from Amon Duul *Live in London* (1973)

3.5 Electronic Sounding Sounds

While we may take the opinion that the electronic synthesizer adds a new range of sounds (albeit simple) to the composer's "pallette", the composer who uses computer-generated sounds exclusively is often disappointed. The

compositional restrictions imposed by computer music can be dramatically demonstrated: choose a popular or traditional melody which has been successfully orchestrated and performed with a variety of instruments. A voice, flute, or guitar performance may be expressive; a piano performance will sound "flat" until it is harmonized, and the computer performance is mechanical and even embarassing to the extent that it will not bear repeated listening. The composer must "throw a lot of notes at the problem" in order to compensate for the extreme acoustic simplicity of this medium, and the "bleeps and beeps" which characterize computer music are not solely the composer's choice. It is common to find jazz bass players and closet heavy metal enthusiasts side by side in making beeps in the computer music studio.

At this point it is appropriate to comment on the "purpose" of computer music, since it is often said that the duplication of traditional instruments is not a worthwhile goal for computer music. This discussion can be approached from two fairly distinct orientations. According to one orientation, computer music should be used to produce novel sounds or sound structures which would be impractical or impossible to attempt using other tools.

Composers are often drawn to computer music for its potential as a performer or realizer of their work, as well as for its possibilities as an electronic medium. This orientation is especially applicable for beginning composers, or when novel compositional techniques are employed and feedback is desired. Computer music may short-circuit the syndrome of the composer who must wait many years for a composition to be discovered and performed for the first time. If the computer is considered primarily as a medium for realization or personal expression, providing the expressive and performance capabilities of traditional instruments is not an inappropriate

goal. Of course, if there existed a computer model which perceptually duplicated the properties of a physical instrument, the model could easily be 'fiddled' to produce novel sounds which exceed the practical or physical limitations of its prototype.

Given the ostensible emphasis of computer music on the production of novel sounds, it is somewhat suprising to note that the fundamental computer sound synthesis techniques, including additive and subtractive synthesis, waveshaping, and frequency modulation, were either pioneered or are commonly employed in analog musical synthesizers. The exceptions to this observation are techniques which process (natural) sounds; linear prediction in particular can produce effects beyond those of the analog vocoder.

It may also be suprising to note that practicing musicians often complain about the sound quality of digital synthesizers:

> "The only way to get any sound that comes close to being considered fat on a digital synth is to have 64 oscillators playing the same thing. With the older analog synthes, you could often use just two oscillators, and the sound had a certain warmth and size that you can't really find now." --Eddie Jobson (keyboard player with Alan Holdsworth and U.K.) [Keyboard 84]

The old electronic oscillators did not stay in tune for very long, and the electronic oscillator may share some of the nonlinear characteristics and "imperfections" of mechanical vibrators [Chamberlin 80, p.489].

3.6 Modelling the Sound versus Modelling the Instrument

The presumption motivating the preceeding discussion is that these imperfections and nonlinear effects are the final product of centuries of design and evolution of musical instruments, and they are the foundation of

the character and expressiveness of the instrument. The original motivation for computer music is surely its unlimited potential, so it is not appropriate at this point to declare computer music to be hopelessly inexpressive.

A major focus of computer music research has been the study of physical instruments and the investigation of modelling methods which might rival the acoustic complexity of physical instruments. Effects identified in acoustic research have been incorporated in the oscillator/function table models. For example, the stretched (slightly inharmonic) spectrum of the piano has been simulated by detuning the oscillators in an additive synthesis model [Schottstaedt 77].

It is interesting to reflect on our knowledge of musical acoustics, and its implications for our modelling strategy. Though we may know how to build a particular physical instrument, the computer modelling of the sound of that instrument requires years of research, and (at present) the resulting model can usually be distinguished from its prototype without difficulty.

An alternative approach is to transfer our knowledge of the instrument's sound from the physical domain to the computer by digitizing the sound. There are several obvious objections to this approach. It entails digitizing all of the pitch, amplitude, and performance combinations which the instrument is capable of producing, because (as was described previously) the sound of a note at a particular amplitude (for example) is generally distinct from an amplified version of a quieter note of the same pitch. A second objection is that there is usually no point in merely digitizing a sound and playing it back without modification. Both of these objections are overcome if the data can be conveniently parameterized in a data reduction procedure such as [Schindler 84]. A third drawback of this approach is that one is dependent

on natural sounds for source material, and that it yields a shallow and specific model (if the parameterized data is considered as such) which may not be easily adapted beyond its original context. The parameterized data does not contribute much to our understanding of the instrument or its sound.

A third modelling approach is to start with the physics of the instrument rather than its sound. This distinction is illustrated in the following equations:

$Y = SIN(WT)$ sine wave (3-1)

$Y'' = -W^2Y$ harmonic oscillator (3-2)

(Y'' denotes the second derivative of Y with respect to time). Both 3-1 and 3-2 result in a sinusoid in Y, but 3-2 suggests a physical model rather than the closed form solution or approximation to that model.

3.7 Models of Physical Vibrators

While computer models of the sound of an instrument are a significant research topic, our knowledge of the physical construction of an instrument may be transferred to a digital model which generates some of the nuances and expressiveness of its prototype. As an example, the vibrating string is modelled by the *wave equation* rather than as an oscillator:

$$Y''(T) = K * Y''(X)$$

($Y''(.)$ denotes the second partial derivative of Y with respect to the parenthesized variable). The constant K is related to the string tension and density.

The closed form solution of the wave equation is well known, but analytical solution becomes difficult as terms describing friction and string stiffness, and other factors are added. An initial formulation of friction is

$Y''(T) = K * Y''(X) - B * Y'(T)$

i.e., B is a damping term proportional to the string velocity. A more accurate formulation of friction accounts for the effect that energy at higher frequencies is damped or dissipated more quickly than at lower frequencies. [Hiller 71] derived an approximation for string stiffness as well as an improved friction term.

In general, physical models take the form of differential or integral equations, which may be implemented digitally by several methods. The author has been working with finite difference implementations of the wave equation for several years, and more recently several stochastic delay-differential equations have been explored. In implementation, physical models are immediately distinguished from oscillator models. For example, the wave equation string model must be tuned by adjusting its tension! (The author was not able to analytically calibrate the a particular value of tension/density constant to a particular absolute frequency, so tuning was a fairly time-consuming process). The harmonic content of the string model also depends on where it is plucked or struck, and how hard. These performance nuances would be tedious to approximate with an oscillator model. Figures 3-2, 3-3 show the outputs of pickups placed at different locations on the string model.

Appendix B is a program listing of the simple stochastic delay-differential equation

$Y = F(T,X(T)) \mid Y'' = X - K1 * Y + K2 * Y_{-D} + K3 * Y'$

where X is a random shock excitation and Y_{-D} refers to the value of Y delayed by time D. This equation resembles a digital filter except that it is nonlinear and does not obey the principle of superposition. When differential equations (such as this one) are invented for the purposes of complex sound

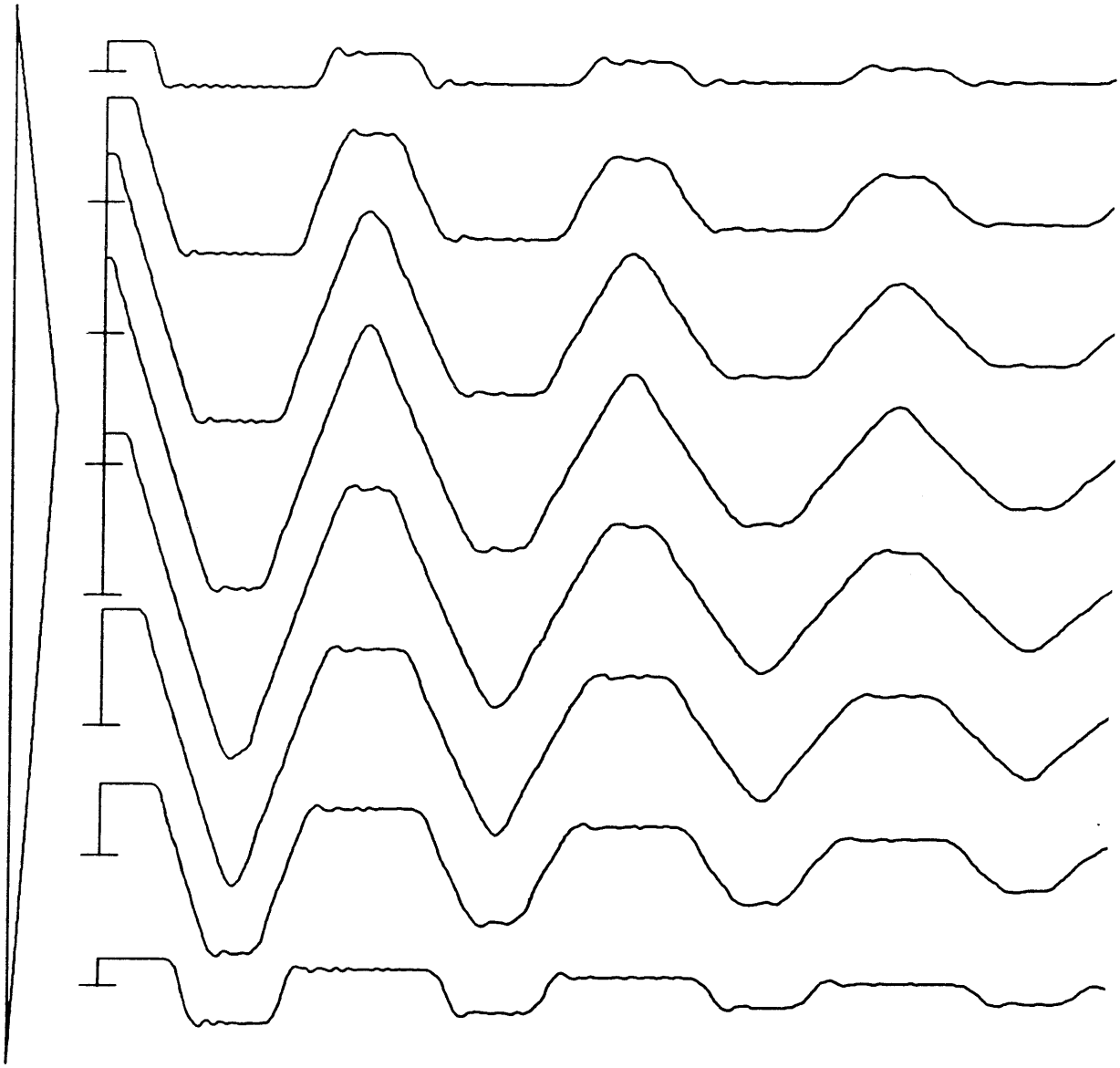

Figure 3-2: Initial string position representing a pluck (displacement as a function of location), and several periods of string motion shown at equally spaced pickups along the string (displacement as a function of time). Damping is exaggerated.

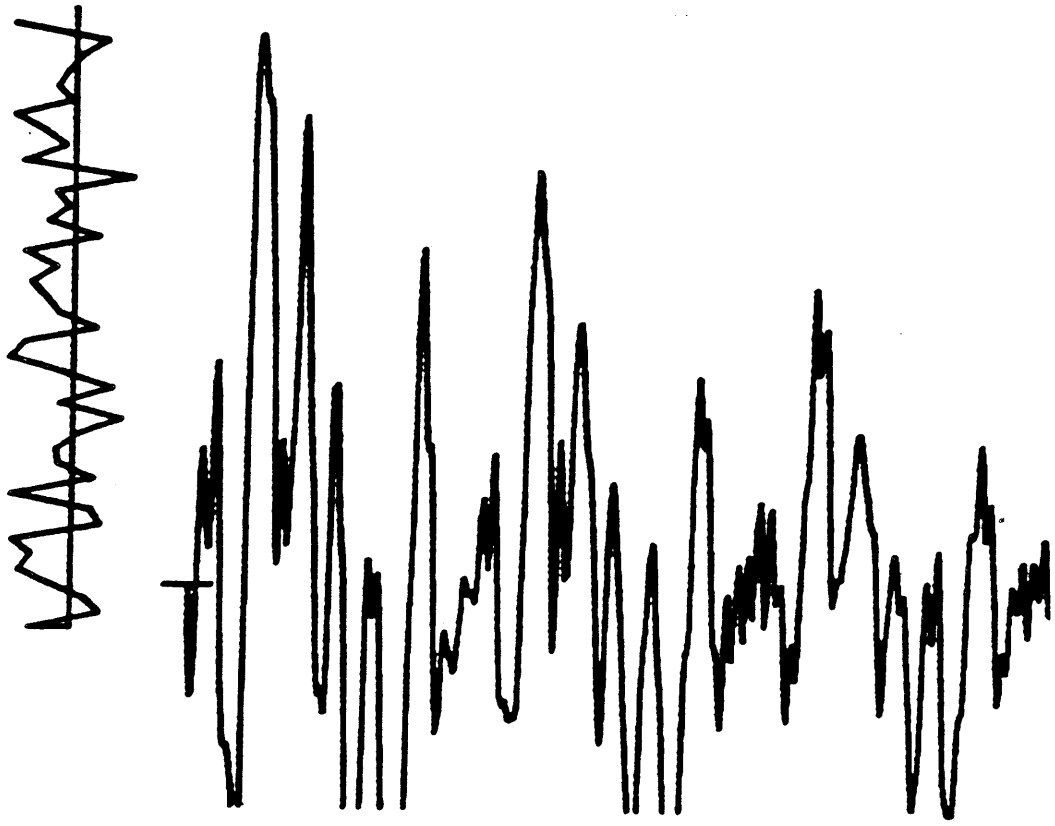

Figure 3-3: Initial string position approximating a metallic sound (displacement as a function of location), and several periods of string motion. Damping is exaggerated.

generation, stability is often a problem, particularly if the equation involves feedback. A solution to this problem (and one which is not inappropriate if the physical implications of the equation are considered) is to insert a limiter or clipper into the feedback portion of the equation. The feedback terms should also be damped, and if the damping is carefully adjusted then the limiter or clipper will only be invoked during portions of each sound event. This may provide a complex sub-event with a perceptual function similar to the attack portion of most physical instruments. [Saaty 81] describes analytical techniques for determining the stability conditions of equations such as this one.

In a listening evaluation, the digital implementation of a non-trivial physical model has several striking qualities. The author's string model, which accounts (to a greater or lesser extent) for air friction, string stiffness, resonant coupling, and the resonance of a sounding board, does not duplicate or even strongly suggest the sound of a plucked string! While the bridge coupling and sounding board are poorly modelled at this point, the complexity and elusiveness of such a perceptually simple sound as a plucked string has been quite suprising.

On the favorable side, the string model does not sound particularly like an electronic synthesizer, and the performance nuances and acoustic complexity of the model are sufficient to sustain a simple or even monophonic score. Some of the acoustic features identified in working with this model have been approximated with an oscillator/wave table model. The resulting 'instrument' has been able to "support" music written for piano, thus contradicting the dictum that the computer is such a distinct medium that it will not support music written for other instruments. On the contrary, it is the author's assumption that the computer is at least potentially the universal medium.

With few exceptions, physical models have not been used in computer music. The major computer languages do not contain all the necessary programming constructs (conditional statements, arrays, and iteration) to permit the implementation of these models, and the design of these languages was guided by the belief that the physical modelling approach is too expensive. This belief was unarguable ten years ago, when one could hope to afford a few minutes of CPU time per day. At present, the computational resources available in a computer graphics lab (for example) can usually offer several hours of equivalent CPU time per day. The physical modelling approach is not "hopelessly inefficient" at this point; in fact, the harmonic oscillator implemented by finite differences is the most efficient ways of generating a sinusoidal oscillation other than by table lookup.

The author's computer music composition *Ossature Metallique*, performed at the spring 1984 computer music concert at MIT, contained two instruments which utilized finite difference equatons. Most of this composition was synthesized using a computer music language developed by the author. The development of this language was prompted by the lack of portability of the existing computer music languages (Musics 10 and 11 are written in PDP-10 and PDP-11 assembler languages), as well as to provide a tool for exploring synthesis methods.

3.8 A Script-driven Multiprocess Programming System

The architecture of this computer music language was shaped by the requirements that it be portable and provide a 'open-ended' programming environment comparable to existing programming languages. Facilities for the basic synthesis techniques, including additive synthesis, frequency modulation, moving average and autoregressive filters, waveshaping, and

linear prediction would be provided, but the major capability of the language would be its programming constructs. Returning to Example 1, notice that the implementation of this instrument consisting of several OSCILs in a (non-music) programming language is only a matter of several table lookups and incremented indices:

```
            /* OSCIL  */
value  = table  [index];
index  = index  + freq  mod  tablesize;
return(value);
```

With the exception of linear prediction, the other standard synthesis techniques are also fairly simple to program.

The major task in writing a computer music language is the evidently the development of the language interpreter or compiler, and the writing of an interpreter or compiler for a new language is a task which should be measured in man-years. Given our requirement that the new music language should provide a programming environment comparable to a programming language, it made a lot of sense to start with an existing programming language, and modify it for sound generation. The conversion of a language compiler or interpreter for a particular machine conflicts with the desire for portability, however.

Portability is achieved if we develope a language translator whose *output* is an existing higher-level language. The music language will be translated into a common programming language and then compiled. The language translator is easily (if not ideally) implemented using a macro processor. The language Ratfor, which is translated into Fortran with a macro processor, provides a precedent for this approach. The source for a version of the M4 macro processor was available and used for this purpose [Kernighan 76]. The disadvantage of implementing the translator with a general-purpose macro

processor is that language-specific error checking is not possible. The music language has enough syntax requirements beyond that of the underlying programming language to justify the term "language", and errors in the music language are not detected until run time in the current implementation.

Much of this 'syntax' is determined by the additional requirements the sound synthesis process be driven by a score file, and that the instruments be reentrant. Once an orchestra (collection of instruments) has been designed and compiled, a particular composition which uses that orchestra is described in a score file such as was shown in Example 2. It is particularly desirable that an instrument be capable of acting with several sets of parameters 'simultaneously' (in virtual time) to generate concurrent sound events such as chords. This capability will be termed 'reentrant'. Unfortunately the most common programming languages do not have the multiprocessing and scheduling constructs to directly support these features.

Abstracting from the musical context, the facility of a system providing script-driven control of multiple processes in time is equally applicable in computer animation. The implementation and use of this facility in a script-driven computer animation system is described in the next chapter.

Chapter Four

The Visual Environment

In this chapter, the graphical marionette figure and the objects comprising its environment are united in a script-driven animation system named Fish. Some of the vocabulary describing the system reflects its origin in the computer music language described in the previous chapter.

The system consists of a script interpreter written in a symbolic interpreted language such as Lisp, an orchestra manager which oversees the animation process, and instruments which accomplish the animation rendering. The instruments are written in a compiled language such as Pascal or PL/1 for efficiency. For practical reasons (specifically, the limitations of the local Lisp/PL/1 interface) the animation is accomplished as two passes.

4.1 The Script Interpreter

The "Lung" script interpreter reads and evaluates the script to produce a script output file. For example, the script input

```
(Repeat 10
        (setq name (gensym))
        (makemove Curtime name 1.0
                (pickrts bounds) (pickdrts drtsbounds))
        (@ Curtime)
                (mary fletcher scale (?in 0.5 1.2))
)
```

generates twenty lines of script output in which the position and movement direction of ten figures (rendered by the MARY program) are chosen at random. The sizes of the figures are also chosen at random, in the range 0.5..1.2 of the default size.

The Lang program contains relatively few preprogrammed constructs: the power of the program is that of the Lisp language itself. Any script input which is not identified as valid input to the animation system is assumed to be a Lisp program fragment and is evaluated. If the Lisp code occurs within a known script statement, the result of the evaluation is substituted into the statement. Lisp code occuring outside of script statements is evaluated for side effects. The predefined constructs can thus be grouped in macros or extended by defining new functions (Appendix D).

4.2 Script-Driven Animation

The scripting facility of this animation system defines a *declarative/procedural* interface, in which the declarative form of the script is converted to drive the procedural (programmed) animation rendering software. Thus, no programming is needed to develop an animation. The scripting process is further simplified by the use of spatial digitizers to directly specify motion (described in the next chapter).

The orchestra manager partially implements a multiprocessing metaphor, in which multiple copies of any instrument may be simultaneously active. Unfortunately the widely available computer languages do not support multiprocessing. In this system multiprocessing is implemented by imposing standard calling conventions (at the procedural level) on instruments and by use of a macro processor. In the resulting environment one instrument or renderer can be active in as many instances as desired, regardless of the extent of the static (impure) data required by each instance. No global data structures are suggested, so a variety of distinct approaches to computer rendering may be present in a single animation. This is evident in the "Nick's grove" animation, where cloud (particle system) figures stroll

through an environment represented by polygonal facets, and are observed by an invisible camera (Figure 4-1).

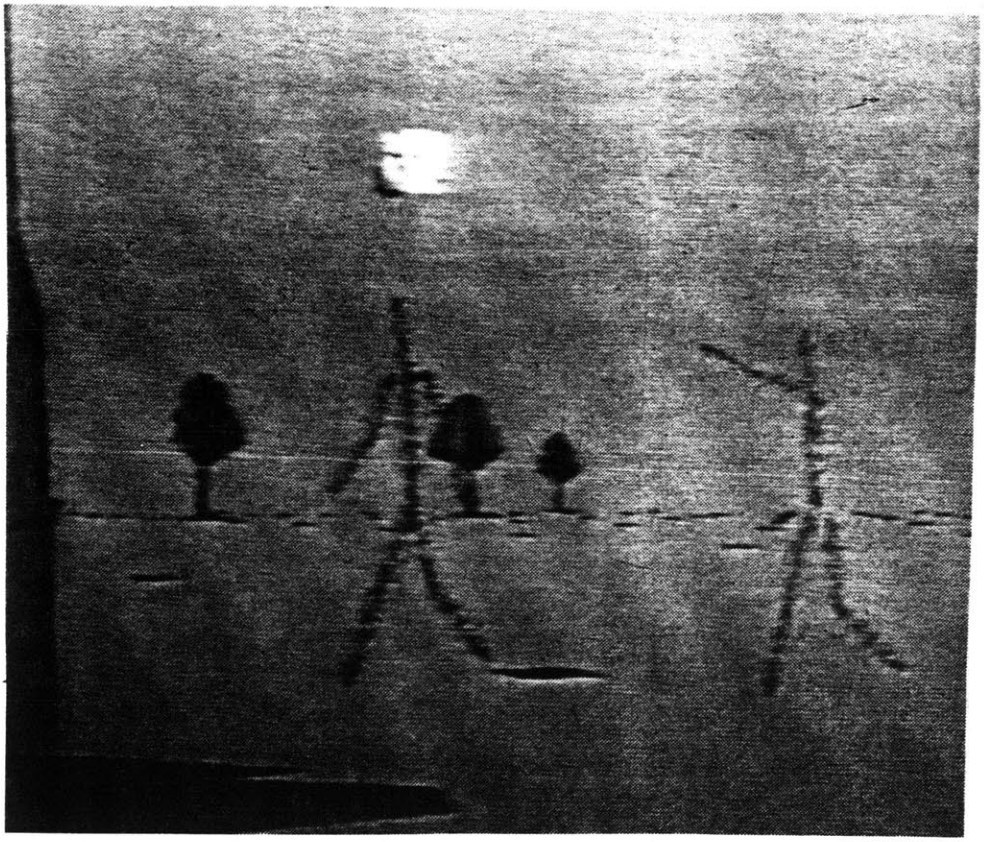

Figure 4-1: A frame from the "Nick's Grove" animation. The falling object is a head which is about to drop onto the shoulders of the first figure.

A user's view of some of the features of this animation system is presented below, with reference to the standard terminology of ASAS for comparison [Reynolds 82].

Motions fall inbetween the ASAS concepts of Newton and Actor. They are called once per frame, in a consistent but unspecified order. As the name implies, motions typically define motion, but they are also an appropriate

mechanism for specifying any quantities which change during the course of the animation. For example, a camera zoom could be accomplished by creating a motion to change the camera's focal length.[3]

Motions are created and destroyed by script statements or by instruments. The required parameters are a name, start time, delta time, starting value, and delta value. The values may be literal or symbolic scalars or vectors. A typical script motion statement is:

```
(move riseandspin Curtime 1.0 vector(1 0 -1) upabit)
```

In this example, the motion RISEANDSPIN defines an interpolated vector quantity which changes from (1 0 -1) at time CURTIME by the amount UPABIT (a predefined vector) during one second.

Motions are origined relative to other positions or motions. This feature facilitates object interactions by relative or coordinated movement (the "message board" system described below is also used for this purpose). For example, the primary figure in an animation sequence might be identified as such by keeping the camera stationary relative to the (moving) figure. A more sophisticated example is the "hungry beetles" chase, in which each of several "beetles" chases the next beetle, starting from a distant position (Figure 4-1). The size, orientation, and location of a group of objects may optionally be changed by changing an origin common to the motions of the objects.[4]

Instruments are similar to the Actor type in ASAS: when the instrument

---

[3]The current camera program requires its focal length to be a literal.

[4]Whether this happens is currently a compile-time parameter; in some cases it may be desirable to change the size and orientation of an object independently of any other objects which are origined relative to it.

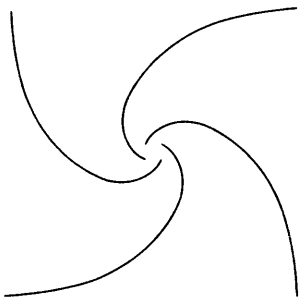

Figure 4-2: Diagram of the spiral paths generated by four "hungry beetles" starting at the vertices of a square (from [Abelson 81)]

instance is active, it will be awakened once each frame and provided with the local data space of that instance. The instrument executes and then returns control to its caller.

This view of the instrument as a somewhat autonomous entity which executes (effectively) in parallel with other processes is subtly but distinctly different from the traditional architecture in which control resides in a single program which invokes a hierarchy of subroutines. Reynolds calls the instrument's world view "object oriented" (other definitions for this term have been proposed).

The order in which instruments are called is defined by coordinates which are updated and broadcast by each instrument instance. Usually the instrument is responsible for rendering a particular type of object at a particular place, and the coordinate is the distance of the object from the camera. Typically the place is defined as a motion instance. Instruments need not do rendering, so they provide a way to perform once-per-frame tasks which are not served by the motion concept. In fact, the 'motion server' is itself such an instrument, with all of the motions as its instances.

Instruments are activated by instrument calls in the script (typically) or by other instruments. The instrument call consists of an activation time, the name of the instrument, and parameters to the instrument. Each instrument defines its own parameter convention, so there is no need to impose a unified parameter structure on diverse instruments. A typical script call, featuring an instrument which is a renderer, is:

```
(@ time1) (bnuts6 riseandspin carrot offset 0 range 300)
```

In this example, at TIME1 an instance of the bnuts6 renderer is created. This instance renders the CARROT object following the motion RISEANDSPIN.

An instrument can deactivate ("unplay") itself, or it can be deactivated by a script statement or by another instrument.

Instruments can communicate with one another by entering their messages in a "message board" database. The utilities which are used for this purpose are the same as those which accomplish the declarative/procedural data transfer and which provide each instrument instance with its local data. A limited example of instrument communication is seen in the "Nick's grove" animation, where Nick's head drops out of a tree and lands on one of the figures (Figure 4-1). This figure is broadcasting the position of its head. The Nick head picks up this position, origins itself relative to it, and moves to the origin. In this way a very precise movement is easily scripted.

4.3 Evaluation

As a system for scripting animation, the Fish has proved fairly robust and flexible. As an animation system, Fish suffers from simplistic rendering and hidden surface algorithms. Like the proverbial four-lane highway, the system has been 'filled to capacity' sooner than expected. Part of the

difficulty here is a local operating system limitation on the size of programs. Currently unneeded instruments must be removed before new instruments can be added.

The applications and results of this system are described in the next chapter. This work was judged to be inappropriate, however, and it was succeeded by the fairly nebulous approach indicated chapter six.

Chapter Five

The Marionette in its Environment

The current work has succeeded in its intended application of portrayal of figures in an environment, as well in an unforeseen application (computer generated holography). Progress and remaining difficulties in the area of body tracking will be described before discussing these applications.

5.1 Motion Specification by Digitization

As originally conceived, the Architecture Machine Group figure modelling project posited a real-time graphic representation of a body-tracked figure [Bolt 81]. A full-body tracker is currently under development but is not operational at this time. In the absence of a body tracker, the motion of the graphical marionette figure has been specified using the walking algorithm, key-frame interpolation, and partial body tracking.

Shigeru Suzuki has adapted a spatial digitizer manufactured by Polhemus Navigational Sciences to provide a partial body-tracking capability [Suzuki 84]. The spatial digitizer returns the position and orientation (azimuth, elevation, and roll) of a hand-held digitizing sensor. Suzuki observed that the spatial digitizer could track a limb (two nodes in the figure model) by making use of the orientation data.

The digitizing sensor is attached to a distal limb segment (e.g. to the forearm). The position and orientation data together define an axis in space parallel to the axis of the distal limb segment. The position of the distal limb

segment is then fixed by considering its position relative to the attached sensor. When the position of the distal limb segment is determined, the proximal limb seement is known to connect a stationary body node to the proximal node of the distal limb segment.

Inaccuracies in the spatial digitization cause the proximal limb segment to stretch or compress. This difficulty is overcome when the limb position is converted to the articulation tree representation. The position of the proximal node of the distal limb segment (= the distal node of the proximal limb segment) is used to determine the articulation matrix rotation angles for the proximal limb segment, and the absolute position of this node is replaced with the correct length of the proximal limb. Similarly, the absolute position of the distal limb segment is discarded after rotational angles are determined. The articulation tree walk will then generate a correctly proportioned figure, and the position of the body-tracked limb will deviate from its prototype only as a result of digitization error.

The author augmented the figure model to allow the motion of one or more of its limbs to be specified by partial body tracking while the figure as a whole is moved by the walking algorithm or by keyframe interpolation. Programs to direct the figure's walking and the motion of objects in the environment using a digitizing tablet (the "posit" program in Appendix C) and by algorithmic methods (Appendix D) were also implemented.

What initially appeared to be minor technical difficulties in this body tracking effort were found to be serious or even crippling. When the spatial digitizer is used as an interactive pointing or positioning device, the digitization noise is easily compensated by the operator. When it is used as a tracking device, however, the digitization noise causes the figure motion to appear mechanical or insect-like.

Several attempts at filtering the noise have not entirely eliminated it. The magnetic operation of this digitizer is disrupted by the graphic display, so that the digitized space is not only nonlinear but non-monotonic. The author would like to propose that simply reducing the amplitude of the noise (with a low order filter) will not be sufficient for the body tracking application, because the residual low-amplitude, high-frequency noise stands out in observation. Human visual perception appears to be quite sensitive to discontinuities in velocity as well as in position. An attempt to splice body tracking motion to key-frame or algorithmic motion would need to consider the discontinuity in velocity at the splice; this problem was judged to be beyond the time scale of this thesis.

Another problem is the limited tracking space (approximately 2-4 cubic feet) of the digitizer. When combined with the restriction that only one limb may be tracked, the result is that only some fairly mundane gestures can be input. The integration of a body-tracked limb motion in figure movement generated by the walking algorithm does not always appear natural. In our experience, the body tracking application is distinct from spatial digitizing and requires specialized tools.

Perhaps the biggest drawback of the current body-tracking effort is the absence of a real-time display device. The turn-around time between tracking and display is several minutes due to the filtering, and the subsequent frame-by-frame display portrays a sequence of positions rather than a motion. The real-time display of a stick figure is well within the capability of cheap vector displays, but the body-tracking (triangulation) and filtering tasks on the full body will require dedicated computation. A new body tracker using one or more dedicated microprocessors is currently being developed by Tetsuo Semba.

5.2 Computer-Generated Figure Animation

While the development of a full body tracker is still underway, several animations depicting figures moving and interacting in virtual three-dimensional environments have been produced.[5]

Michael Roper's animation "Concrete Jungle" juxtaposes the stark, geometrical character of the graphical marionnete and its environment with the complexity and warmth of several streetside types, captured on video. This animation is a setting of the song by Bob Marley.

Jennifer Hall, an artist at the MIT Center for Advanced Visual Studies, produced an animation which uses computer graphics to successively expand the viewer's perceptions beyond the inevitable attempt to assign a conventional (real world) spatial interpretation. The initial appearance of figures walking upside-down suggests that the camera is upside down, but the appearance of upright figures in the same scene brings the realization that the other figures are "walking on the ceiling". A shift in the camera's position reveals that the initial scene was only one "wall" of a larger space.

Sarah Dickenson's animation places the graphical marionette figures against an abstract, industrial backdrop inspired by the artist Leger. This animation makes extensive use of single-limb body tracking to effect a "mechanical ballet".

The "Nick's Grove" animation by Michael Teitel and Jennifer Hall was described in the previous chapter.

---

[5]These animations are well under a minute in length, due to limitations on computer time (the more complex scenes in these animations required approximately two-hours of computer time to produce each second of animation).

5.3 Computer-Generated Holograms

A system for producing lenticular holograms has been constructed at the Polaroid corporation. Approximately one-hundred photographs of a scene are obtained at equally spaced intervals as a camera moves from right to left across the scene. The resulting hologram is monochromatic and has parallax in the horizontal direction only [Benton 83].

The virtual camera and regular sampling of an animation system such as Fish make it an appropriate tool for producing a computer-generated lenticular hologram. The virtual camera is panned across the imaginary scene at the speed required to obtain the desired number of frames and the desired perspective shift.

Several computer-generated holograms have been produced using Fish. These are not the first computer-generated holograms of this type, but they appear to be the first computer-generated holograms to incorporated medium-resolution, shaded imagery.

5.4 Conclusion

While the scope of the Architecture Machine figure modelling project is partially paralleled by work at the University of Pennsylvania, the Computer Graphics Research Group at the Ohio State University, and particularly at the Simon Fraser University, the previously-described animations depicting the motion and action of multiple figures in imaginary environments appear to be unique at this time.

It is hoped that this work will provide the basis for further devlopment. Priorities in the future will continue to include the development of a full-

body tracker. The aquisition of a real-time picture system will allow immediate feedback in body-tracking and and enable interactive *scripting-by-enactment* [Bolt 81]. A secondary goal is the development of the figure model to provide facial expressions, speech, and better overall representation.

Appendix A

PotatoSlice manual

This appendix is the users' manual for the PotatoSlice programs described in Chapter 2.

A.1 Overview

Suppose you are creating an object which you will call potato". The following (annotated) sequence of commands will run the programs to allow you to input and view an object described by parallel planar slices. The commands themselves are distinguished in **bold** type.

1. Type **cwd >u>course>potato**. Files will be created in your directory though the programs are located in this directory.

2. Type **editslice potato** to create the object. Remember to specify y values in increasing order, and to enter slices counterclockwisely. The first and last points on a particular slice are assumed to be connected, so it is not necessary to 'close' a slice by entering two points at the same location.

3. Type **brunch potato**. This runs various number crunching programs and prints out mathematical-looking things.

4. Type **Bnuts6 potato**. Bnuts6 is the program for producing a perspective, shaded display of the object. A minimal command sequence for using Bnuts6 is: **init trans 0 0 1000 go**

A.2 Display Commands

The commands accepted by the Bnuts6 display program are described below. The starred commands are toggles, i.e., if their current value is FALSE then the command sets them to TRUE:

init	resets the perspective transformation matrix, including any previous translations, rotations, or scaling commands.
rx,ry,rz	rotate the object about x, y, or z axes
scale <x y z>	scale the object in three directions
trans <x y z>	translate the object
* flipshade	if flipshade is true a solid rendering is performed, otherwise a wire-frame is shown. flipshade is initially false.
* gourd	specifies smooth (Gouraud) shading
* outline	causes tiles to be outlined with jaggy black lines
* backfacing	causes back-facing tiles to be shaded red (initially FALSE)
light <x y z>	specifies three components of the direction vector to the light source. The z component should be negative if you want to see anything. Light must be specified before the flipshade command is given.
writefunc	This allows the rendering of translucent objects and other effects. Try 'writefunc avg'. 'writefunc ?' lists the command abbreviations. See 'help ram$param' also.
saveposition	saves the rotations/translations/scales for the current position in a file in your directory
setposition	retrieves a position saved earlier
newfile	opens a new object file

go begins the rendering

Bnuts6 uses the left-handed coordinate system:
The x-axis extends in the positive direction to the right on the screen.
The y-axis extends in the positive direction up on the screen.
The z-axis extends in the positive direction into the screen.

Therefore, to view a normal object you should do the rotations first, and then translate the object away from the origin into the viewing space. This usually includes a large positive (e.g. 500) translation in z.

A.3 Other Programs

bnuts7 & nuts7 are like (B)nuts6 except that the 'gourd' shading command produces a primitive texture mapping rather than Gouraud shading. To run these programs, specify

```
datapath:              >u>username
p218file               try pam240 or sin240
object                 the name of the object
```

It is necessary to give the 'ox' and 'ampl' commands before translating and displaying the object. A sample command sequence is:

```
init
ox
ampl 60
rx 0.3 ry 0.3 rz 0.3
trans 0 0 700
gourd go
```

The texturing process sometimes exceeds the grayscale range (0-255). To fix this, do (after quitting the program):

```
linear
range? 0 330
initial color? 0 0 0
final color? 1 1 1
```

There are several variations on the brunch program:

brunch	is very fast & usually acceptable. Bnuts6 & bnuts7 can display objects produced using 'brunch'. Try things out with brunch first, then run crunch5 if desired.
crunch	is slow for large objects ('bonehead' takes about an hour if you are the only person on the machine). crunch does 'global optimization' and it produces slightly better results than brunch. crunch objects are displayed with nuts6 & nuts7.
crunch4	like crunch, but displayed with Bnuts6 or bnuts7. This program is recommended over crunch. Slow for large objects.
crunch5	crunch5 produces the best results, especially for objects which slant (successive contours/slices are horizontally shifted). Slow for large objects. Use Bnuts6,bnuts7 to view.
	Several other useful programs:
reflector	reflects each slice about the first point in the slice. Use this for making objects with one axis of symmetry, eg a head.
countiles	counts the number of tiles (triangles) in an object
sysfilter	this program low-pass filters the screen image. Try it.
TSRS.doc	documentation of the object data structure

Appendix B

Program for a stochastic differential equation

A program listing of a finite difference implementation of an example stochastic differential equation:

```
DDE:proc;     /* stochastic delay-difference equation example */

/* derivation..
y'' = -ky     harmonic oscillator
y'' = -k1y + k2y'     w/ friction
y'' = -k1y + k2y(-t) + k3y'   w/ feedback
y'' = x -k1y + k2y(-t) + k3y' w/ random shock excitation

notation, y+ = y(t+1), dt=1
y+ - 2y= + y- = x -k1y= + k2y-t + k3y= - k3y-
y+ = x -k1y= + 2y= + k3y= - y- - k3y- + k2y-t
y+ = x + (2 - k1 + k3)y= - (1 + k3)y- + (k2)y-t -k1

k1   force
k2   feedback
k3   friction

retime
y= = (2 - k1 + k3)y- - (1 + k3)y-2 + (k2)y-t-1

*/
call ioa(" for speech-like signal, try 0.01, 0.005, -0.001, 10 ");

dcl F [1:512] flt;
dcl i fix;
dcl memory[0:20] flt;  dcl Mlen fix init(20);
do i=0 to Mlen; memory[i]=0.0; end;
dcl (k1,k2,k3) flt;
dcl (iMlen,iMlen1,iMlen2,oMlen) fix;
dcl y flt;
```

```
dcl x flt;


l:
call askn("excitation force,feedback,friction constants,pitch period ? ",k1);
call askn("feedback ? ",k2);
call askn("friction ? ",k3);
call askn("pitch period > ",p);
dcl (p,ip) fix; ip=0;

excit = SL$oof();      /* correlated random 1/f↑2 excitation */

iMlen = 3;
do i=1 to 512;
     ip = ip + 1;   if ip>p then do;
                          ip=0;
                          call ioan(".");
                          x = SL$oof();
                    end;
     iMlen1 = iMlen - 1;    if iMlen1 < 0 then iMlen1 = iMlen1 + Mlen;
     iMlen2 = iMlen1 - 1;   if iMlen2 < 0 then iMlen2 = iMlen2 + Mlen;
     oMlen = iMlen + 1;     if oMlen > Mlen then oMlen = 0;


/*    y= = (2 - k1 + k3)y-  - (1 + k3)y-2  + (k2)y-t-1 */
     y = (2.0 - k1 + k3) * memory[iMlen1];
     y = y - (1.0 + k3) * memory[iMlen2];
     y = y + k2 * memory[oMlen];
     y = y + x;
     memory[iMlen] = y;
     F[i] = y;

     iMlen = iMlen + 1;     if iMlen > Mlen then iMlen = 0;
     x=0.0;
end;

goto l;

end ;
```

Appendix C

FISH manual

Several important preexisting instruments for generating the marionette and its environment are described here (see Chapter 5). In the descriptions, "P<N>" refers to the instruments' Nth parameter, and "&OPT" means that the remaining parameters are optional (defaults are assumed) and may be specified in any order.

cursor — A 3d coordinate system/cursor which is useful for cheaply representing objects while their motions are defined. It draws x,y,z axes in red,green,blue. The default size is 100. If P3 is not a number it is ignored.

P2 = motion
&OPT P3 = size

grid — A rectangular grid of square "spots" which can be used to represent a planar surface in space.

P2 = motion
&OPT xmin,xmax,zmin,zmax,dx,dz,color,spotsize

bnuts6 — The renderer for PotatoSlice objects (see Chapter 2 and Appendix A).

P2 = motionid
P3 = object
&OPT gourd, plate, wire,
outline, ↑outline range offset

hack — This is the "graphical marionette" stick figure. The figure may be rendered by line segments or by "clouds". Several types of clouds have been developed. Particle system clouds composed of a distribution of one-pixel "particles" are perceptually integrated in the eye but result in fairly

strong aliasing during movement. Gaussian clouds remove the high spatial frequencies of particle clouds but practically restrict the animation to monochrome if a 8- or 9-bit picture display is used.

P2 = motion
P3 = template file (fletcher)
&OPT
cloud particle field clouds
gcloud Gaussian clouds
stick stick figure
scale (default size is 100 pixels)
range range in color matrix:
offset..offset + range-1
offset offset in color matrix; default is 255
radius 'fatness' fraction: 1.0 generates a
normally proportioned figure; 1.5 is a fat
figure
track fname node speed

The track command replaces the Cutting 'walking algorithm' data with data for a limb obtained with Shigeru's Polhemus body tracking data. <fname> is the file name of the tracking data. The extension .trackdata is assumed. <node> is the node where the tracked motion is attached. It replaces the walking data at that node & its son. nodes 10,13 are left,right shoulders in the fletcher template. <speed> is a floating speed; 1.0 = real time (as it came from the Polhemus).

Because the stick figure moves differently than other objects (e.g. it does not move at a constant velocity), it should be scaled using the scale parameter rather than (as is usually done) by scaling a motion.

billboard

Calls polymap, to simulate placement of a 'billboard' in space. This instrument was written by Michael Teitel. A sample script line:

0 billboard motionid picname <options>;

options:

crop (x1 y1 x2 y2 x3 y3 x4 y4)

Describes the boundaries of the billboard in a pic file in Ramtek coordinates (0,0 = top left; 639,439 = bottom right). If entered in clockwise order beginning with the top left, the billboard will start oriented as seen with loadpic

otherwise x1 y1 will be top left
x2 y2 top right
x3 y3 bot right
x4 y4 bot left

clear (percent) percent between 0 and 1 (0 = transparent; 1 = opaque, lower than .2 seems useless).

fade (time start finish)

time = time2fade
start = start percent
finish = finish percent

zorg (z) z is the z coordinate of the origin for rotations. (billboards are origined at the center of the picture in x and y)

Appendix D

Functions for randomized movement

The control constructs provided in Lung (Chapter 4) can be extended by defining new functions. The Lung **(incl <filename>)** command will load Lisp program fragments so that they may be used in a script:

```
(setq rtsbounds '(-1000 1000  0 0   0 2000))
(setq drtsbounds '(-30 30 0 0 -30 30))
(incl "randommoves")
;tell mary fletcher to stagger for 10 seconds
;(stagger name time dur srate:nsteps/sec
;         rtsbounds drtsbounds frac)
(stagger (gensym) 0.0 10.0 5 rtsbounds drtsbounds 0.2)
(dclR mary)
(mary stagger fletcher gcloud)
(end)
```

These sample functions illustrate the definition of a staggering motion and motion in a random direction:

```
(defun stagger (name starttime dur nsteps-sec rtsrange drtsrange-sec frac)
;
(prog (time dt startpos dir speed)
    (setq startpos (pickrts rtsrange))
    (setq speed (?in (/$ (veclen drtsrange-sec) 2.0) (veclen drtsrange-sec)))
    (setq dir (vecnormalize (pickrts drtsrange-sec)))
    (setq dt (/$ 1.0 (float nsteps-sec)))
    (setq speed (*$ speed dt))     ;assume drtsrange-sec in time 1.0
                                   ;now speed is per-sample magnitude
    (cond ((flolessp dt SPERIOD) (error "toomany steps")))
    (setq time starttime)
    (make time 'move name dt 'rts (append '(0 0 0) startpos)
                          'drts (vec->drts (vecscale dir speed)))
a   ;(gc)
    (setq time (+$ time dt))
    (setq dir (vecdriftxz dir frac))
```

```
        (make time 'move name dt 'drts (vec->drts (vecscale dir speed)))
        (cond ((flolessp (+ $ time dt) (+ $ starttime dur)) (go a)))

        (setq time (+ $ starttime dur))
        (cond ((flolessp Maxtime time) (setq time Maxtime)))
))      ;stagger



(defun rndmove (rtsrange drtsrange-sec)
;generate a straight line motion in a random direction
;initial position is chosen within rtsrange
;speed is chosen within drtsrange-sec
(prog (name)
        (setq name (gensym))
        (setq Moves (cons name Moves))
        (putprop name t 'Move)
        (putprop name (picrts rtsrange) 'Start)
        (putprop name (picrts drtsrange-sec) 'Delta)
(return name)
))      ;rndmove
```

[Abelson 81]
H. Abelson and A. diSessa.
Turtle Geometry.
MIT Press, 1981.

[Badler 79a]
N. Badler, J. O'Rourke, and H. Toltzis.
A spheroidal representation of a human body for visualizing movement.
Proc. IEEE 67(10):1397-1403, October, 1979.

[Badler 79b]
N. Badler and S. Smoliar.
Digital representations of human movement.
Computing Surveys 11(1), March, 1979.

[Badler 80]
N. Badler, J. O'Rourke, and B. Kaufman.
Special problems in human movement simulation.
Computer Graphics 14(3), 1980.

[Badler 84]
N. Badler.
What is required for effective human figure animation?
In *Graphics Interface.* 1984.

[Benade 76]
A. H. Benade.
Fundamentals of Musical Acoustics.
Oxford University Press, 1976.

[Benton 83]
S. Benton.
Photographic holography.
SPIE Journal 391, 1983.

[Boissonat 81]
J. D. Boissonnat and O. D. Faugeras.
Triangulation of 3D objects.
In *Proceedings of the 1981 International Joint Conference on Artificial Intelligence*, pages 658-660. 1981.

[Bolt 79]
R. Bolt.
Spatial Data-Management.
Technical Report, MIT Architecture Machine Group, March, 1979.
DARPA Report

[Bolt 80]
R. Bolt.
"Put-That-There": voice and gesture at the graphics interface.
Computer Graphics, August, 1980.

[Bolt 81]
R. Bolt.
Proposal for the development of a Graphical Marionette.
Technical Report, MIT Architecture Machine Group, 1981.

[Brennan 82]
S. Brennan.
Caricature Generator.
Master's thesis, MIT, August, 1982.

[Calvert 82]
T. W. Calvert, J. Chapman, and A. Patla.
Aspects of the kinematic simulation of human movement.
IEEE Computer Graphics and Applications 2(3), November, 1982.

[Chamberlin 80]
H. Chamberlin.
Musical Applications of Microprocessors.
Hayden, 1980.

[Christianson 76]
H. Christianson and T. W. Sederberg.
Conversion of complex contour line definitions into polygonal element mosaics.
Computer Graphics 13(2):187-192, August, 1976.

[Cutting 78]
J. E. Cutting.
A program to generate synthetic walkers as dynamic point-light displays.
Behavior Research Methods and Instrumentation 10(1):91-94, 1978.

[Dooley 82]
M. Dooley.
Anthropometric modelling programs -- a survey.
IEEE Computer graphics and applications 2(3), November, 1982.

[Ekman 73]
P. Ekman.
Universal facial expressions in emotion.
Studia Psychologica 25(2):140-146, 1973.

[Ekman 75]
P.Ekman.
Facial Affect Scoring Technique (fast): A First Validity Study.
Semiotica , 1975.

[Fetter 82]
W. Fetter.
A progression of human figures simulated by computer graphics.
IEEE Computer Graphics and Applications 2(3), November, 1982.

[Foley 82]
J. D. Foley and A. Van Dam.
Fundamentals of Interactive Computer Graphics.
Addison-Wesley, 1982.

[Fuchs 77]
H. Fuchs.
Optimal surface reconstruction from planar contours.
Communications of the CACM 20(10), October, 1977.

[Ganapathy 82]
S. Ganapathy and T. G. Dennehy.
A new general triangulation method for planar contours.
Computer Graphics (Siggraph 1982) 16(3):69-75, 1982.

[Ginsberg 83]
C. Ginsberg.
Human body motion as input to an animated graphical display.
Master's thesis, MIT, May, 1983.

[Gouraud 71]
H. Gouraud.
Continuous shading of curved surfaces.
IEEE Transactions on Computers 20(6):623-628, June, 1971.

[Hatze 81]
H. Hatze.
Myocybernetic Control Models of Skeletal Muscle.
University of South Africa Press, 1981.

[Herbison-Evans 78]
D. Herbison-Evans.
NUDES 2: A numeric utility for displaying ellipsoid solids.
Computer Graphics 12(3):354-356, August, 1978.

[Hiller 71]
L. Hiller and P. Ruiz.
Synthesizing musical sounds by solving the wave equation for vibrating objects.
Journal of the Audio Engineering Society 19(7), July/August, 1971.

[Hutchinson 60]
A. Hutchinson.
Labanotation.
Theater Arts Books, 1960.

[Kay 77]
A. Kay and A. Goldberg.
Personal dynamic media.
Computer, March, 1977.

[Keppel 75]
E. Keppel.
Approximating complex surfaces by triangulation of contour lines.
IBM Journal of Research and Development 21:2-11, January, 1975.

[Kernighan 76]
B. W. Kernighan and P. J. Plauger.
Software Tools.
Addison-Wesley, 1976.

[Keyboard 84]
Keyboard staff.
The power and glory of lead synthesizer.
Keyboard 10(3), February, 1984.

[Kingsley 81]
E. C. Kingsley, N. Schofield, and K. Case.
SAMMIE-a computer aid for man-machine modelling.
Computer Graphics 15(3):163-169, August, 1981.

[Lewis 82]
J. Lewis.
Computer simulation of human line drawing.
Unpublished paper

[Lewis 84]
J. Lewis and P. Purcell.
Soft Machine: a personable interface.
In *Graphics Interface.* 1984.

[Mathews 69]
M. V. Mathews.
The technology of computer music.
MIT Press, 1969.

[Maxwell 82]
D. Maxewll.
Caricature Generator.
Master's thesis, MIT, August, 1982.

[Menowsky 82]
J. Menowsky.
Video graphics and grand jetes: choreography by computer.
Science 82 3(4), May, 1982.

[Negroponte 79]
N. Negroponte.
Talking heads--display techniques for persona.
Technical Report, MIT Architecture Machine Group, 1979.
Unpublished paper.

[Negroponte 80]
N. Negroponte, A. Lippman, and R. Bolt.
Transmission of presence.
Technical Report, MIT Architecture Machine Group, 1980.
Proposal to the Cybernetics Technology Office, DARPA

[Negroponte 81]
N. Negroponte.
Media Room.
In *Society for Information Display, Proceedings*, Volume 22. 1981.

[Nisselson 83]
J. Nisselson.
Model Kit.
Master's thesis, MIT, June, 1983.

[Nitchie 79]
E. Nitchie.
How to Read Lips for Fun and Profit.
Hawthorne, 1979.

[O'Rourke 80a]
J. O'Rourke.
Image Analysis of Human Motion.
PhD thesis, University of Pennsylvania, 1980.

[O'Rourke 80b]
J. O'Rourke and N. Badler.
Model-based image analysis of human motion using constraint propagation.
IEEE Trans. PAMI 2(6):522-536, November, 1980.

[O'Rourke 81]
J. O'Rourke.
Triangulation of minimal area as 3D object models.
In *Proceedings of the 1981 International Joint Conference on Artificial Intelligence*, pages 664-666. 1981.

[Oppenheimm 75]
A. Oppenheim and R. Schafer.
Digital Signal Processing.
Prentice-Hall, 1975.

[Parke 72]
F. Parke.
Computer generated animation of faces.
In *Proceedings of the ACM Annual Conference*, Volume 1. 1972.

[Parke 82]
F. Parke.
Parameterized models for facial animation.
IEEE Computer Graphics and Applications 2(3), November, 1982.

[Pearson 76]
K. Pearson.
The control of walking.
Scientific American 235(6), December, 1976.

[Pierrynowski 82]
M. Pierrynowski.
A Physiological Model for the Solution of Individual Muscle Forces during Normal Locomotion.
PhD thesis, Simon Fraser University, 1982.

[Platt 81]
S. M. Platt and N. I. Badler.
Animating Facial Expressions.
Computer Graphics (Siggraph 1981) 15(3), August, 1981.

[Rabiner 78]
L. Rabiner and R. Schafer.
Digital Processing of Speech Signals.
Prentice-Hall, 1978.

[Rashid 80]
R. Rashid.
Lights: A system for the interpretation of moving light diplays.
PhD thesis, University of Rochester, 1980.

[Reynolds 82]
C. W. Reynolds.
Computer animation with scripts and actors.
Computer Graphics 16(3):289-296, 1982.

[Rongo 82]
R. Rongo.
Robotic Vision Systems, Inc. Data-Base Users' Manual..
Robotic Vision Systems (formerly Solid Photography), 1982.

[Saaty 81]
T. L. Saaty.
Modern Nonlinear Equations.
Dover, 1981.

[Schindler 84]
K. Schindler.
Dynamic timbre control for real-time digital synthesis.
Computer Music Journal :46-50, Spring, 1984.

[Schottstaedt 77]
B. Schottstaedt.
The simulation of natural instrument tones using frequency modulation with a complex modulating wave.
Computer Music Journal :46-50, November, 1977.

[Schwartz 84]
R. Schwartz, Y. Chow, S. Roucos, M. Krasner, and J. Makhoul.
Improved hidden Markov modelling of phonemes for continuous speech recognition.
In *International Conference on Acoustics, Speech, and Signal Processing.* IEEE, 1984.

[Shannon 49]
C. Shannon.
Communication in the presence of noise.
Proc. Inst. Radio Eng. 37(1):10-21, January, 1949.

[Stearns 75]
S. D. Stearns.
Digital Signal Analysis.
Hayden, 1975.

[Stumpf 90]
C. Stumpf.
Tonpsychologie.
S. Hirzel, 1890.

[Suzuki 84]
Shigeru Suzuki.
Body tracking by spatial digitization.
Technical Report, MIT Architecture Machine Group, 1984.

[Weil 82]
P. Weil.
About Face.
Master's thesis, MIT, August, 1982.

[Weizenbaum 64]
J. Weizenbaum.
ELIZA--A computer program for the study of natural language communication between man and machine.
Communications of the ACM 9(1), 1964.

[Zeltzer 82a]
D. Zeltzer.
Representation of complex animated figures.
In *Graphics Interface.* 1982.

[Zeltzer 82b]
D. Zeltzer.
Motion planning task manager for a skeleton animation system.
Technical Report, Siggraph Tutorial, 1982.