MAC-TR-61 (THESIS)

INTERACTIVE COMPUTER-MEDIATED ANIMATION

by

Ronald M. Baecker

June 1969

Project MAC

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

This blank page was inserted to preserve pagination.

INTERACTIVE COMPUTER-MEDIATED ANIMATION

by

RONALD MICHAEL BAECKER

S.B., Massachusetts Institute of Technology
(1963)
M.S., Massachusetts Institute of Technology
(1964)

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF
PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF
TECHNOLOGY
June, 1969

Signature of Author Ronald Baecker
Department of Electrical Engineering
March, 1969

Certified by Edward O. Olson
Thesis Supervisor

Accepted by _____
Chairman, Departmental Committee
on Graduate Students

INTERACTIVE COMPUTER-MEDIATED ANIMATION

by

Ronald Michael Baecker

Submitted to the Department of Electrical Engineering in March, 1969 in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

ABSTRACT

The use of interactive computer graphics in the construction of animated visual displays is investigated.

The dissertation presents a process called interactive computer-mediated animation, in which dynamic displays are constructed by utilizing direct console commands, algorithms, free-hand sketches, and real-time actions. The resulting "movie" can then be immediately viewed and altered.

The dissertation also describes a special kind of interactive computer-mediated animation that exploits the potentialities of direct graphical interaction. The animator may sketch and refine (1) static images to be used as components of individual frames of the movie, and (2) static and dynamic images that represent dynamic behavior, that is, movement and rhythm. Because these latter pictures drive algorithms to generate dynamic displays, the process is called picture-driven animation.

Each representation of movement and rhythm determines critical parameters of a sequence of frames. Thus, with a single sketch or action that generates or modifies the representation, the animator can exercise dynamic control over an entire interval of the movie. One natural way to do this is by mimicking in real time a movement or a rhythm, using a stylus or a push-button.

These concepts are supported by experience with three special-purpose picture-driven animation systems which have been implemented and used on the M.I.T. Lincoln Laboratory TX-2 computer.

The dissertation also presents an outline of the proposed design of a multi-purpose, open-ended, interactive Animation and Picture Processing Language. APPL is a conversational language which accepts direct sketches, direct console commands, and algorithms that control interactive dynamic displays.

Solutions are presented for the following problems: How can the system be structured so that the command set can easily be augmented by the animator? How can movie time be represented in the language, and how does the choice of representation interact with the flow of program and system control? What computational data structure can facilitate the modeling of sequential and hierarchic structures of pictures and dynamic data? How can we provide a rich picture description capability in the language? How can we facilitate the construction of programs which describe the user's interaction with the system?

APPL programs are included to demonstrate that the language can be gracefully used to construct dynamic displays, to build system tools that aid the construction process, and to implement special-purpose interactive computer-mediated animation systems.

Thesis Supervisor: Edward L. Glaser
Title: Associate Professor of Electrical Engineering, M.I.T.
(currently Chairman, Department of Information and Computer Sciences, Case Western Reserve University)

PREFACE AND ACKNOWLEDGMENTS

Several friends have asked me, with perhaps a tinge of envy in their voices, how I was fortunate enough to find a thesis topic that, but for the inexorable pressure from a certain draft, could have been fun.

The thesis began with an illusion of inspiration and my good fortune in being in Boston at a time when interest in computer animation was high. Two ideas kept me awake one night in November of 1966--one was so specific that it was never used, the other so general that the dissertation barely scratches its surface. Yet a false Eureka is better than none, and he who thinks he has found a promising trail has at least found a trail. At that time talks with Wally Feuerzeig and Ted Myer at Bolt, Beranek, and Newman had already awakened my interest in computer animation. Professor Seymour Papert at M.I.T. suggested work on a language for animating stick figures. Professor Ivan Sutherland, now at the University of Utah but then at Harvard, proposed use of waveforms to define changing picture parameters, and Tim Johnson of M.I.T. enlarged upon this suggestion. Group 23 of Lincoln Laboratory had already offered use of the TX-2 computer with its superb graphics hardware and software, a system with which I had become familiar during a summer's work on the interface of graph theory and graphics. So, late in December of 1966, I began work on a stick figure animation program.

By early March a lone stick figure, headless but optimistic, took its first faltering steps across a TX-2 scope. Soon Eric Martin, from the Harvard Carpenter Center for the Visual Arts and Cambridge Design Group, Inc., and I were beginning to learn how to control its dynamics by crudely sketching and resketching waveforms. Thus we began <u>interactive computer-mediated animation</u>.

Somewhat earlier, I had begun intensive and for me thought-provoking discussions on computer animation with Professor Edward L. Glaser of M.I.T., who since then has become Chairman of the Department of Information and Computer Sciences at Case Western Reserve University in Cleveland. In May of 1967 I submitted a thesis proposal, with Professor Glaser as mentor. The proposal stressed the synthesis of movement through the sketching and graphical editing of waveforms, and the creation of an animation language in which pictures could be implicitly described and retrieved in terms of their "properties."

After a summer spent on graphical debugging, I began in October the design of an interactive animation language. There soon followed a premature and abortive attempt at an implementation of the yet fuzzily-defined language. In response to the pressure of a speaking commitment to a group of artists, I returned to the TX-2 to generate more examples of computer animation. I grafted from the stick figure system (ADAM) an abstract dynamic art program (EVE). Having just devised the p-curve as an alternate approach to the specification of continuous movement, I designed EVE to illustrate graphically the power of the dynamic mimicking of a motion and its recording by the computer for use in an animated display. (In this goal I apparently succeeded, for one New York avant-garde film-maker, excited by some animation fragments from EVE, responded, "Man, that's pure underground--pure underground!")

I also constructed a simplified animation system in which one could sketch components of frames. This appealed to the artist in Eric in a way that the other two systems had not. I then realized the need to couple the richness of dynamic control afforded by the use of waveforms and p-curves with the richness of static imagery to which artists and animators were accustomed. The ability to use waveforms and p-curves to define the translation of arbitrary sketched

images, now part of the animation system, was clearly insufficient. Eric continued to stress, and I continued to resist, the notion that animation required discrete picture change as well as continuous movement. What troubled me was an apparent conceptual gap between continuous movement and discrete picture change, and so I returned to the drawing board for another pass at the language design.

Suddenly, in late spring of 1968, the alternation between practical computer animation attempts and conceptual language design began to pay off, just as Professor Glaser had anticipated all along. A solution began to gel, one in which there were nice parallels between continuous and discrete picture change, between continuous and discrete movement, between continuous pictorial operations such as translation and discrete pictorial operations such as insertion and deletion, and between continuous picture attributes defining coordinates and discrete picture attributes defining structure. A viewpoint was crystallized in which certain classes of data sequences, called <u>global descriptions of dynamics</u>, were used as characterizations and generative definitions of ongoing picture change, and in which classes of static and dynamic pictures were used as definitions and synthetic tools for manipulating global dynamic descriptions.

There was sufficient advance on two fronts, throughout the summer and fall of 1968, so that the thesis of the dissertation, stated in the Introduction, could be supported. First, the animation system quickly grew into something quite different and of considerably more power, the <u>GENE</u>ralized-cel Animation <u>SYS</u>tem (GENESYS). As crocodilesses began to cavort across the TX-2 screen, <u>picture-driven animation</u> became a practical reality. Second, the <u>A</u>nimation and <u>P</u>icture <u>P</u>rocessing <u>L</u>anguage (APPL) was refined through the painful construction and testing on paper of algorithms that defined

both dynamic displays and special-purpose interactive computer-mediated animation systems.

And now the dissertation is complete. Many individuals, including those already cited, have contributed to the progress of the research. It is with deep appreciation that I acknowledge:

Professor Edward L. Glaser, for his encouragement, his wise and patient counsel, his insight, and his contagious enthusiasm and faith in the importance of computer animation;

Dr. William R. Sutherland of MIT Lincoln Laboratory, for his constant support and expressed confidence in my work, his many careful readings of chapter drafts, and our numerous thought-provoking discussions;

Professor Murray Eden, for his kind advice and guidance throughout my years at MIT, and his valuable comments on the manuscript;

Mr. Eric Martin, for his freely-given time and spirited interest in discussing and experimenting with animation systems; (by the way, Eric, the system is beginning to look as if it might possibly soon be almost crash-free and bug-proof, I hope...);

Mr. James E. Curry of Adams Associates, for his thoughtful suggestions on parts of the dissertation;

Mr. Timothy Johnson of MIT, Dr. Amedio Armenti of Lincoln Laboratory, and Professor Adolfo Guzman of MIT, for their comments on the manuscript; Mr. Jack Nolan and Mr. Robert Davis of Lincoln Laboratory, and Professor Thomas G. Stockham, Jr., for advice and assistance;

The Digital Computers Group(23) of MIT Lincoln Laboratory, for staff services, computer time, and stimulating environment of many talented professionals; Specific contributions have come from Mr. C. S. Lin, who designed the circuit linking the movie camera to the computer, and Mr. Robert McCormack and Mrs. Nancy Johnson of Adams Associates, who programmed parts of GENESYS;

Mr. Ephraim Cohen, Mrs. Johnson, Miss Barbara Koppel, and Miss Lynn Smith, for their generation of experimental mini-cartoons, and for their helpful comments on GENESYS;

Mrs. Lila Hartmann, for her typing of the final copy;

ADAM, EVE, and GENESYS, for assistance in preparing the illustrations found in the dissertation; it is believed, incidentally, that all movies from which frames are included would obtain one of the following two ratings under the Motion Picture Code of Self-Regulation, either G, suggested for GENERAL audiences, or M, suggested for MATURE audiences (parental discretion advised);

Mr. Robert Goldberg, Miss Patricia Lewis, my sister, and my parents, for their help in numerous ways, at numerous times.

I also wish to thank the others, nameless but not forgotten, who have helped.

Any errors remaining in the manuscript are of course solely my responsibility.

The work reported herein was supported in part by Project MAC, an MIT research project sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract NONR-4102(01), and in part by MIT Lincoln Laboratory, operated with support from the U.S. Advanced Research Projects Agency.

TABLE OF CONTENTS

INTERACTIVE COMPUTER-MEDIATED ANIMATION

Using Interactive Computer Graphics in the Construction of Animated Visual Displays

Page

ABSTRACT 2

PREFACE AND ACKNOWLEDGMENTS. 4

TABLE OF CONTENTS. 9

WHAT IS ANIMATION? 19

INTRODUCTION
What has work to date in computer animation achieved, and what has not been achieved? The general features of digital computer-based animation systems, and two unique systems using analog and hybrid computers, are described. The approach of the dissertation is stated, and distinguished from what has been done in the past. The thesis of the dissertation is then formulated. . . 21

A NOTE TO THE BUSY READER. 33

Footnotes to Introduction 34

I.A. INTERACTIVE COMPUTER-MEDIATED ANIMATION. 36

1. THE ROLE OF DIRECT GRAPHICAL INTERACTION IN THE SYNTHESIS OF ANIMATED VISUAL DISPLAYS 37

2. INTERACTIVE COMPUTER-MEDIATED ANIMATION
The process of interactive computer-mediated animation is defined in terms of the minimal components required to realize a working system.. 46

3. A SCENARIO ILLUSTRATING THE USE OF AN INTERACTIVE COMPUTER-MEDIATED ANIMATION SYSTEM 52

4. IMPLICATIONS OF THE SCENARIO
The scenario is interpreted and its significant features listed. 60

Footnotes to I.A.. 62

Page

I.B. THE DEFINITION OF PICTURE DYNAMICS--PICTURE-DRIVEN ANIMATION 64

Three old approaches to the synthesis of a sequence of frames are described:

1. THE INDIVIDUAL CONSTRUCTION OF EACH FRAME IN THE SEQUENCE 65

2. THE INTERPOLATION OF SEQUENCES OF FRAMES INTERMEDIATE TO PAIRS OF CRITICAL FRAMES 69

3. THE GENERATION OF FRAMES FROM AN ALGORITHMIC DESCRIPTION OF THE SEQUENCE. . . 70

A new approach, picture-driven animation, is introduced:

4. PICTURE-DRIVEN ANIMATION
Pictures "drive" algorithms to generate a sequence of frames. The pictures represent data sequences, whose successive elements determine critical parameters in each frame. The data sequences are called global descriptions of dynamics. 72

5. THE RELATIONSHIP, IN GENESYS, OF CELS AND CEL CLASSES TO GLOBAL DYNAMIC DESCRIPTIONS
GENESYS is a picture-driven animation system in which algorithms produce dynamic displays by combining static images, called cels, with global dynamic descriptions . . . 74

There are three kinds of dynamic descriptions:

6. PATH DESCRIPTIONS
Picture changes that are potentially continuous variations of value are expressed by path descriptions 76

7. SELECTION DESCRIPTIONS (1)
Picture changes that are recurring discrete choices of pictures, data, events, or actions are expressed by selection descriptions.. 80

8. RHYTHM DESCRIPTIONS
Rhythm descriptions express patterns of the triggering, pacing, coordination, and synchronizing of picture change. 82

Page

9. SELECTION DESCRIPTIONS (2)
--DYNAMIC HIERARCHIES
The hierarchic construction of complex movements out of simple ones is described. 85

10. GENESYS GENERALIZED
Additions to GENESYS are contemplated. These would allow dynamic descriptions to play an even greater role in defining picture dynamics. 87

Footnotes to I.B.. 93

I.C. THE ART OF DEFINING AND REFINING GLOBAL DYNAMIC DESCRIPTIONS--THE USE OF PICTORIAL REPRESENTATIONS. 95

1. SPECIFYING GLOBAL DYNAMIC DESCRIPTIONS
Six general approaches are described 96

2. WAVEFORMS, P-CURVES, AND OTHER PICTORIAL REPRESENTATIONS OF PATH DESCRIPTIONS
Both static pictures, such as the waveform, and dynamic pictures, such as the p-curve, are considered.. 99

3. PICTORIAL REPRESENTATIONS OF PATH DESCRIPTIONS--AN ANALYSIS
Criteria are established for evaluating the utility of various pictorial representations in the construction and modification of dynamic descriptions.. . . . 107

4. SKETCHING AND RESKETCHING PATH DESCRIPTIONS
This includes both the free-hand sketching and dynamic mimicking of desired animated behavior.. 113

5. EDITING AND GRAPHICALLY REFINING PATH DESCRIPTIONS
The essential features of a flexible editing system are outlined. 118

6. SPECIFYING SELECTION AND RHYTHM DESCRIPTIONS
Techniques are similar to those used in the definition of path descriptions.. . . 123

Page

7. COORDINATING PARALLEL ACTIONS
This is a critical problem of picture-driven animation.. 127

8. IMPLICATIONS OF THE FACT THAT DEFINING AND REFINING DYNAMIC DESCRIPTIONS IS AN ART
The argument develops the need for an extensible animation system.. 130

Footnotes to I.C.. 134

I.D. A DETAILED EXAMPLE OF PICTURE-DRIVEN ANIMATION--MANY SYNTHESES OF A PULSATING HEART
Numerous techniques for constructing this simple dynamic display are presented. Each technique facilitates a certain kind of control over the movement and rhythm of the resulting movie. 136

Footnotes to I.D.. 147

I.E. EXPLORATORY STUDIES IN PICTURE-DRIVEN ANIMATION
There are presented results, observations, and interpretations from the use of the three picture-driven animation systems that have been implemented on the M.I.T. Lincoln Laboratory TX-2 computer.. 148

1. CONSTRUCTING, VIEWING, AND FILMING MOVIES ON THE TX-2
Features of the movie construction, playback, and filming processes, common to all three systems, are described. 149

2. ADAM
This is a special-purpose system for the animation of a stick man.. 151

3. EVE
This is a special-purpose system for the animation of an abstract figure. 159

4. GENESYS--THE GENERALIZED-CEL ANIMATION SYSTEM
The process of using GENESYS is further clarified.. 162

Footnotes to I.E.. 174

Page

I.F. CONCLUSION--THE REPRESENTATION OF DYNAMIC INFORMATION--THE CONCEPT OF A DYNAMIC DISPLAY
Picture-driven animation is summarized and its advantages characterized. To exploit these advantages, the animation system must allow a plastic relationship among global dynamic descriptions, pictures, and processes of picture construction (actions)--they must be interchangeable representations of dynamic behavior and equivalent components in the generation of dynamic displays. Thus, further motivation for the design of an extensible animation system is developed. 176

II.A. A GENERAL INTRODUCTION TO THE DESIGN OF AN OPEN-ENDED, MULTI-PURPOSE, INTERACTIVE COMPUTER-MEDIATED ANIMATION SYSTEM. 184

Aspects of the design are motivated in terms of the relevant history of computer graphics and animation:

1. THE ESSENTIAL FEATURES OF SKETCHPAD, BEFLIX, AND CAFE. 185
2. SOME SPECIFIC ISSUES RAISED BY BEFLIX, CAFE, AND GENESYS.. 192

The system must be open-ended--the directly executable commands fundamental to any interactive computer graphics system must also serve as statements in a complete programming language, one which is capable of definitional extension. This conversational language will be called APPL, an Animation and Picture Processing Language:

3. ESSENTIAL FEATURES IN THE DESIGN OF APPL. . 195
4. A SCENARIO ILLUSTRATING SOME USES OF APPL . 199
5. IMPLICATIONS OF THE SCENARIO
The system is flexible and multi-purpose, for it may be used:
(1) to construct animated visual displays, by directly designating a sequence of APPL commands for immediate execution, or by writing a program in APPL;

Page

(2) to build system tools that aid the construction process; and,
(3) to implement special-purpose interactive computer-mediated animation systems 208

6. WHAT IS TO COME, AND WHAT IS NOT TO COME, AND WHY
The language is developed in the next four chapters; the philosophy of the presentation is stated here.. 210

Footnotes to II.A.. 212

II.B. THE INCORPORATION OF A COMMAND LANGUAGE INTO AN ANIMATION PROGRAMMING LANGUAGE 214

1. COMMANDS AND PICTORIAL DATA IN APPL
The interaction of a user with APPL results in the creation of a pictorial data base consisting of structured collections of pictures and numbers.. 215

2. EXTENDING THE COMMAND STRUCTURE
Arbitrary command sequences may be saved for delayed execution under system-directed flow of control. New commands so defined may be nested to arbitrary depth; hence the language is open-ended. 219

3. THE SCOPE ISSUE
The issue of the scope and binding of names in programs is discussed. A distinction is made between the one global (system-wide) pictorial data base, and various local data bases belonging to individual programs. 222

Footnotes to II.B.. 225

II.C. THE INTERACTIVE DEFINITION OF DYNAMIC DISPLAYS IN APPL. 226

1. THE INDIVIDUAL CONSTRUCTION OF EACH FRAME IN THE SEQUENCE
The use of APPL in the individual sketching of a sequence of frames is illustrated. 227

Page

2. THE GENERATION OF FRAMES FROM AN ALGORITHMIC DESCRIPTION OF THE SEQUENCE--THE CONCEPT OF MOVIE TIME
An example of the algorithmic definition of a dynamic display is presented and analyzed, with particular concern for the implications of real time construction and playback of the movie. The discussion motivates the introduction of the concept of simulated, or movie time. 230

3. THE MODULAR DEFINITION OF CONCURRENT DYNAMIC ACTIVITIES
A mechanism of program control, quasi-parallel processing, is adopted from simulation languages to facilitate the specification of parallel actions in a movie. 236

4. THE FLOW OF CONTROL AMONG INTERACTIVE APPL PROGRAMS
Control is distributed, during the passage of movie time, among instances of APPL programs in quasi-parallel execution. A mechanism called the agenda mediates the flow of control. Particular emphasis is placed on the effect one program can have on another, and on ways the animator's use of the stylus, push buttons, and other devices can affect the flow of control and the passage of movie time.. . . 241

Footnotes to II.C.. 251

II.D. APPL'S DATA STRUCTURE FOR MODELING PICTURES AND GLOBAL DYNAMIC DESCRIPTIONS. . . . 252

1. DESIGN CRITERIA FOR A DATA STRUCTURE
The goals are related to the needs for sequential and hierarchic structures in animation and picture processing. 253

Page

2. THE AGGREGATE
This data structure shares certain features with (ordered) sets, arrays, and rings, and is a generalization of all ordered hierarchic data representations. Linguistic constructs for its manipulation are enumerated. The assignment of names to values and the issues of structure sharing are discussed. 256

3. AN ILLUSTRATION OF MODELING COMPLEX PICTURES BY PICTORIAL AGGREGATES. 267

4. ILLUSTRATIONS OF MODELING GLOBAL DYNAMIC DESCRIPTIONS BY SCALAR AGGREGATES 270

The parallel mechanisms for representing structured pictures and dynamic descriptions allow a very plastic representation of dynamic information. It also facilitates the implementation, through extensions of APPL, of the mechanisms required for picture-driven animation:

5. PICTURE-DRIVEN ANIMATION IN APPL. 272

Footnotes to II.D.. 276

II.E. MECHANISMS FOR DESCRIBING AND MANIPULATING STRUCTURED PICTURES IN APPL 277

1. ATTRIBUTES AND PROPERTIES OF PICTURES
An attribute is a scalar-valued or picture-valued function of a picture. The value of an attribute of a picture is called a property. Subpictures possessing certain properties may be isolated by the application of a picture selection function. 278

2. THE DEFINITION OF NEW PICTURE TYPES
The type attribute distinguishes classes of pictures with individual characteristic features. The user may extend the descriptive capability of his language by defining new picture types.. 287

Page

3. AUXILIARY DEFINITIONS OF PICTORIAL ATTRIBUTES AND OPERATIONS
He further extends his ability to manipulate classes of pictures by defining new attributes, and by making auxiliary definitions of attributes and operations which refer to their behavior when applied to pictures of a new type.. 296

Footnotes to II.E.. 302

III.A. SUMMARY AND CONCLUSIONS. 303

III.B. DISCUSSION AND SUGGESTIONS FOR FUTURE WORK
This chapter further comments upon and criticizes the dissertation, points out what has not yet been accomplished, and suggests what should next be done in interactive computer-mediated animation. . . . 307

1. DIFFICULTIES ENCOUNTERED IN IMPLEMENTING AND USING ADAM, EVE, AND GENESYS
These problems are described with reference to published literature on components of the TX-2 interactive graphics environment. 308

With respect to picture-driven animation and the proposed design of APPL, the development of future interactive animation systems is considered, and some unanswered questions are posed for future research.

2. WHAT TO DO NEXT IN PICTURE-DRIVEN ANIMATION. 320

3. THE IMPLEMENTATION AND FURTHER GENERALIZATION OF APPL 321

4. HOW SUPPORTING SUBSYSTEMS, BOTH HARDWARE AND SOFTWARE, COULD BETTER FACILITATE INTERACTIVE COMPUTER-MEDIATED ANIMATION
Some avenues of research towards better environments for interactive animation are suggested. 324

Page

5. THE FEASIBILITY OF INTERACTIVE COMPUTER-MEDIATED ANIMATION
A number of factors will determine the economic and practical viability, present and future, of interactive computer-mediated animation. 328

6. APPLICATIONS OF INTERACTIVE COMPUTER-MEDIATED ANIMATION
Uses in educational filmmaking, psychology, psychiatry, and the arts are discussed. 332

Footnotes to III.B. 335

REFERENCES 337

APPENDIX 0: MULTIPLE-CHOICE EXAMINATION 349

BIOGRAPHICAL NOTE. 350

WHAT IS ANIMATION?

***Animation is not the art of DRAWINGS-that-move
but the art of MOVEMENTS-that-are-drawn.

***What happens between each frame is more important
than what exists on each frame.

***Animation is therefore the art of manipulating
the invisible interstices that lie between frames.

The interstices are the bones, flesh and blood of the
movie, what is on each frame, merely the clothing.[1]

Norman McLaren (*1)
National Film Board--Canada

This dissertation may be regarded in part as an investigation of the use of the computer in "the art of MOVEMENTS-that-are-drawn," in the manipulation of "the invisible interstices that lie between frames."

All footnotes (*...) may be found at the end of the Chapter in which they occur. (*1) is located on page 34.

Animation is the graphic art which occurs in time. Whereas a static image (such as a Picasso or a complex graph) may convey complex information through a single picture, animation conveys equivalently complex information through a sequence of images seen in time. It is characteristic of this medium, as opposed to static imagery, that the actual graphical information at any given instant is relatively slight. The source of information for the viewer of animation is implicit in picture change: change in relative position, shape, and dynamics. Therefore, a computer is ideally suited to making animation "possible" through the fluid refinement of these changes.[4]

Eric Martin

Carpenter Center for
the Visual Arts,
Harvard University,
and
Cambridge Design Group, Inc.

INTRODUCTION

The animation industry is ripe for a revolution. Historical accidents of available technology and knowledge of visual physiology have led to the evolution of the animated film as "one that is created frame-by-frame."[3] The prodigious quantities of labor required for the construction of twenty-four individual frames per second of film have led to a concentration of animation activity in the assembly-line environments of a few large companies, an artificial yet rarely surmountable separation of the artist from his medium, and extravagant costs.[2] (*2) In conjunction with other trends in American society, the result is usually what the English critic Stephenson describes as "the respectable sadism and sterotype of commerce."[3] Yet he offers this hopeful prediction in concluding his 1967 study, Animation in the Cinema: "There seems every reason to look forward to changes which would make it possible for the creative artist to put on the screen a stream of images with the same facility as he can now produce a single still picture."[3] This paper explains how a creative artist, aided by a computer, can define a stream of images with the same facility as he can now produce a very few still pictures.

Although the computer's entrance into animation has been a recent one (1964),[5,6] the growth of interest and activity has been phenomenal.[8-14] (*3,*4) Experience to date strongly

suggests that the following statements are true:

(1) The animated display is a natural medium for the recording and analysis of computer output from simulations and data reduction, and for the modeling, presentation, and elucidation of phenomena of physics, biology, and engineering.[15-20] Depiction through animation is particularly appropriate where simultaneous actions in some system must be represented. If the animation is the pictorial simulation of a complex, mathematically-expressed physical theory, then the film can only be made with the aid of a computer.

(2) The computer is an artistic and animation medium, a powerful aid in the creation of beautiful visual phenomena, and not merely a tool for the drafting of regular or repetitive pictures.[21-25]

(3) The formal modeling of pictures by complexes of algorithms and data facilitates the continued modification of a single animation sequence and the production of a series of related sequences.

This dissertation is a study of ways in which man, aided by a computer in an interactive graphical environment, can synthesize animated visual displays. I stress the coincidence of the words "interactive" and "graphical"; the animator must interact directly with the production of his film in a way which may truly be called graphical, that is, by sketching and

modifying pictures, and by viewing in real time successive versions of the movie. (*5)

Computer animation is the process of constructing animated films using a computer. The approach of this dissertation differs significantly from that of past (and, to the best of my knowledge, present) work in computer animation. The difference is reflected in the phrase, "interactive computer-mediated animation." I use this terminology to stress that the computer is a medium with which the animator interacts graphically to create dynamic displays. These may later be recorded on photographic film or video tape, but this is not an essential part of the process.

We begin, therefore, by reviewing other work in computer animation. Central to the design of any animation system is the choice of the style and extent of man-machine communication best suited to potential users of the system. (*6) We shall look at the choice that has been made in several digital, hybrid, and analog computer systems.

Digital Computer Systems

All but two computer animation projects have used general-purpose digital computers, with punched-card or typewriter input of written language, and off-line movie output to a microfilm recorder or scope-mounted camera. The animator generally submits a movie as a deck of punched cards, and views the resulting film hours or days later. CAFE, the Computer-Aided Film Editor system developed at M.I.T. Lincoln Laboratory, allows a movie-maker to define and edit a film from an on-line typewriter.[26] His interaction, however, is not graphical--he cannot sketch pictures and he too must wait hours or days to see the film.

Some workers using digital computers have made minimal changes to existing programming languages, such as FORTRAN, in order to attain more quickly the capacity for movie generation.[10] (*7) In other instances a language has been designed expressly for animation, as was done in providing Bell Telephone Laboratories movie-makers with BEFLIX, the first true animation programming language.[5,6] Users of both augmented FORTRAN and of BEFLIX can in principle write arbitrary algorithmic descriptions of dynamic pictures. Still a third approach is that of CAFE, which is a package of commands, a control language.[26] A limited number of useful algorithms are embedded in the interpretation of the commands. This special-purpose graphics system lacks the ability to accept and interpret algorithms expressed in a programming language. (*8)

The ability to define pictures by giving formal generative descriptions has enabled many programmers, scientists, and engineers to produce animated films. That many such individuals can now become animators is an attractive result.[27] Very few artists, however, have attempted computer animation. (*9) To use BEFLIX or CAFE, an animator trained only in traditional media and techniques is forced to learn a completely new "language," a completely new way of thinking. (*10) Stan Vanderbeek, a creative and inventive film-maker who has experimented with many media, claims that it took him nearly two years to attain real fluency in the use of BEFLIX. (*11) This must imply that BEFLIX, although a significant and pace-setting contribution, fails (for Vanderbeek, at least) to transform the raw computer into an effective, responsive, animation-machine and animation medium. I claim that failure is likely a priori, for some direct graphical interaction is a prerequisite for achieving this transformation. (*12)

Analog and Hybrid Computer Systems

Two other groups achieve varieties of direct graphical communication with very different equipment configurations, hybrid and analog computer systems.

In a 1967 paper, Miura, Iwata, and Tsuda report:[28]

> We have recently developed two computing methods for producing animation: (1) Representing the picture in mathematical equations and moving it by switching the constants of the equations (analog computer method); and (2) having two frames drawn by an animator. The curve indicating the movement between these two frames is then read into the computer. According to the results calculated by the computer, animations between the two frames are machine drawn (hybrid computer method). ...
>
> ...
>
> This method [the analog computer method] has the advantage that the pictures produced on the cathode ray tube can be moved on real time. However, the method also has two important drawbacks. One is the fact that it is difficult to produce a picture which is faithful to the artist's intention. The second drawback is that in order to obtain complicated pictures the computing circuit itself will become complicated.

Furthermore, they note that the constants in the analog computer method may be varied "either manually or under the program control of the digital computer." Yet no techniques for generating and modifying these variations are described. Setting the constants individually in each frame is not a viable method.

In their hybrid computer solution for interpolation; the animator must decompose all sketches into curved segments. He must define, for each segment, the original curve, the curve after it has been moved or modified, corresponding

sequence of representative points along each curve, and paths of interpolation for the two endpoints of the segment. In a variation on the technique, the final curve need not be specified. A combination of translation, rotation, expansion, and contraction is employed by the computer to carry out an interpolation of the representative points. The solution, although the only one published to date, is far from ideal. The decomposition into segments and the identification of corresponding points will be time-consuming and frustrating for an animator, especially since the choice of segments and points must be based on geometrical qualities of the curves and not on symbolic relationships among picture parts. Furthermore, no mention is made of the problem of hidden lines, or occlusion of picture parts.

Lee Harrison of Computer Image Corporation in Denver has designed and built a hybrid computer specifically for animation.[29] The system, not yet described in the published literature, is novel and innovative. Movies with texture may be viewed in real time. Hardware techniques solve a useful special-case of the hidden line problem. All images must be synthesized by literally wiring a plugboard to model a stick figure, and then modulating the frequency, length, and intensity of a vector that spins rapidly around each stick. The result is an interesting shimmering, filmy quality, and so Computer Image terms the process "synthesizing skin."

Harrison is an artist as well as engineer, and has conceptualized and developed his machine over a span of years. Sample experimental films, many produced in a single night's work, are vivid proof of his control over his medium, and constitute the most exciting computer animation produced to date. Yet I question whether many animators will find as natural and useful the enforced modeling of pictures through stick figures and rotating vectors. Wiring a model on a plugboard is archaic in this age, but was quick and economical in the early stages of their work. They are planning to introduce a small digital computer to assume this and other functions. From what I have seen and heard described, they still need better techniques for describing and controlling the temporal behavior of the parameters of the rotating vectors.

These analog and hybrid systems allow direct graphical interaction, but lack fundamental and I feel essential capabilities usually found only in large scale digital computers. What is missing is the capacity for flexible storage and retrieval of numerous images and dynamic descriptions, and the ability to define arbitrary formal and analytic picture generators where appropriate. All but the hybrid approach of Miura, et al, force the animator to build pictures out of a very limited set of analog picture generators. We do not yet know if there exists such a set of generators from which a sizeable portion of interesting pictures can easily be synthesized and modeled. Any picture can be built from points,

or circles under affine transformations, or vectors rotating about stick figures, but the construction may be awkward and the representation inelegant. The hierarchic picture description capability of suitably programmed digital computers enables a flexible and powerful solution to this problem, for it is in principle easier to rewrite a program than it is to rewire a machine. Until we obtain a better understanding of the descriptive capability of limited picture models, experimentation with general-purpose digital computers would appear useful and essential. After we possess such understanding, special-purpose digital and analog hardware should be added, and the digital computer can evolve towards a truly useful and economical animation-machine.

This Dissertation

I have attempted to develop a process that augments harmoniously the techniques with which most animators are familiar, that reflects and extends the ways of thinking to which they are accustomed. The animator can sketch with a stylus on a horizontal tablet. (*13) What he draws appears immediately on a CRT display scope above the tablet.(*14) Both static images and complete movies may be viewed directly on the scope.

An operational system (GENESYS) allows the animator, as soon as he lifts the pen, to sketch frames or components of frames. Within this same intuitive "language" of sketching and mimicking, he can also synthesize picture dynamics, that is, descriptions of movement and rhythm. The techniques at his disposal are unlike those found in any traditional animation medium or existent computer animation system. Part I of the dissertation presents the concepts that are the heart of this approach.

The skilled computer-animator should also be able to grow and modify his own animation-machine. This will be possible once the concepts developed in Part II of the dissertation are embodied in an operational system. The animator will then have the power to define algorithms that extend the basic set of system commands, that augment the basic pictorial constructs and operators. He will himself design and add

mechanisms that synthesize and aid the synthesis of dynamic displays.

To summarize, the dissertation seeks to:

In Part I,

(1) describe the role of direct graphical interaction, sketching, and mimicking in computer animation, resulting in the process we shall call interactive computer-mediated animation; and,

(2) develop a new approach to the specification of picture dynamics, one which exploits the capacity for direct graphical interaction, and which we shall call picture-driven animation; and,

In Part II,

(3) based on the principles evolved in (1) and (2), outline the design of a multi-purpose, open-ended, interactive animation programming system.

These goals are currently best met in the standard interactive computer graphics environment, a general-purpose digital computer, buffered by hierarchies of auxiliary storage, and augmented by hardware for directly displaying pictures and for accepting real time drawn input.

The thesis of the dissertation is that:

(1) Direct graphical interaction is a flexible and powerful aid to the construction of animated visual displays; interactive computer-mediated animation is feasible and deserves further research.

(2) Picture-driven animation is a simple yet powerful, intuitively suggestive process with which to express animated displays, one which successfully exploits

the availability of direct graphical interaction; the important feature of this process is that it enables an animator to define and refine picture dynamics, movement and rhythm, through sketching, mimicking, and graphically manipulating static and dynamic images.

(3) The progression from the ability to sketch, through the ability directly to command the computer to aid sketching and the modification of sketches, to the ability to extend the capabilities of the animation-machine and the characteristics of the animation medium, is natural both for the growth of the system and for the learning experience of the non-programming animator. The last stage of the progression requires an animation programming language which is a multi-purpose, open-ended, picture description language. (*15)

A NOTE TO THE BUSY READER (*16)

If you are interested in computer animation,

(1) Read the Table of Contents, the Introduction, and Chapter I.A.

(2) Skim quickly I.B. through I.E., then reread what interests you.

(3) Read I.F., and as far into Part II as you have time and interest.

(If you have no technical training in computers and programming languages, you should be able to read through I.E., but it will be difficult to go much further.)

(4) Alternatively, obtain and read a copy of my condensation of Part I.[30] Then return to the dissertation.

If you are interested in on-line systems and languages for computer graphics, or in extensible languages,

(1) Execute Step 1 above.

(2) Read quickly Chapter I.B., then skip or skim I.C. through I.E.

(3) Begin reading carefully at I.F.

FOOTNOTES -- Introduction

(*1) McLaren has been termed "a highly individual artist, ..., a research worker in film techniques" (Halas and Manvell,[2] p. 290), "a craftsman and an innovator" (Stephenson,[3] p. 73). His work is described in Halas and Manvell,[2] pp. 290-2, 301-3, and in Stephenson,[3] pp. 69-73.

(*2) I.B.1 contains a brief discussion of aspects of traditional animation. For a fuller account, see References 2 and 3.

(*3) Some interesting speculations on uses of the computer in animation and sculpture appear in Sutro's[7] early paper (1962).

(*4) Reference 8 through Reference 13 are to recent conferences, either devoted partially or exclusively to computer animation.

(*5) The phrase in real time, as it is used in the dissertation, is defined in context in I.A.2, (3) and (4).

(*6) In the "real world," economic constraints play a role in the decision. Fortunately, we have been able in the dissertation to ignore questions of current economic viability. This is fitting, for in the computer field what is expensive today may literally be cheap tomorrow.

(*7) The desire to proceed with the movie-making partially explains the dearth, since BEFLIX, of fundamental research on multi-purpose programming languages for animation.

(*8) In II.A.1 and II.A.2 there is a detailed analysis of BEFLIX and CAFE viewed as languages.

(*9) Reichardt,[25] p. 71.

(*10) Traditional media and techniques include sketching with pen and ink on sheets of celluloid and superimposing them on an animation stand (cel animation), and positioning and moving cut paper on an animation stand. In either case the resulting image is photographed with a camera mounted on the stand; then construction of the next frame begins. Still another method is pixillation, in which the passage of time is artificially accelerated by shooting with a standard movie camera individual frames of live action, spaced at large intervals ranging from seconds to days. See also I.B.1, and Reference 2 and 3.

(*11) This comment was made by Vanderbeek after a recent (late 1968) lecture and film-showing at the Harvard Carpenter Center for the Visual Arts.

(*12) We shall return in III.B to the question of how much direct graphical interaction is required. A related question is how much direct graphical interaction one can afford. Cost, in terms of the kind of films that are made at Bell Laboratories and Lincoln Laboratory, is one major reason that BEFLIX and CAFE are graphically off-line.

(*13) See I.A.2,(3). The stylus is in fact a sheathed ball-point pen, with a wire connecting it to the computer. The tablet is covered by a glass plate, on which paper may be placed if so desired.

(*14) Cathode Ray Tube. See I.A.2,(4). Figures I.A.11-13 depict the scope, stylus, and tablet.

(*15) Throughout the writing of Part II, we adopt the ideal and convenient fiction that the animator himself will master the language described there. The thesis does not stand or fall on this shaky hypothesis. The animator may always need a programmer to write and debug precise statements of algorithms. Nonetheless, the language should be a communications medium for the two of them. The animator should gradually become able to use the language informally and to comprehend roughly the meaning of programs. For, as we shall stress often, the artist who wishes to exploit fully the capabilities of the computer medium must in some sense understand its unique feature, the ability to execute with ease algorithms specified in some language. It is part of the thesis of the dissertation that the language developed in Part II is well conceived to achieve this more limited goal.

(*16) We are indebted to Adolfo Guzman for the concept of the busy reader. Guzman,[31] p. 14.

I.A. INTERACTIVE COMPUTER-MEDIATED ANIMATION

One thing only is needed for the pictorial narrator--a knowledge of physiognomics and human expression. After all, he must create a convincing hero and characterize the people he comes into contact with; he must convey their reaction and let the story unfold in terms of readable expressions. Does this not need a skilled artist who has spent years drawing from plaster casts, who has drawn those eyes, ears, noses which, as Töpffer says [the humorist and draughtsman Rudolph Töpffer of Geneva, in a pamphlet on physiognomics published in 1845], are the pleasant exercises which art schools impose on budding artists? For Töpffer all this is waste of time. The practical physiognomics needed for a picture story could be learned by a recluse who never sets eyes on any human being. All he needs is drawing material and some perseverance. For any drawing of a human face, however inept, however childish, possesses, by the very fact that it has been drawn, a character and an expression. This being so, and being quite independent of knowledge and of art, anybody who wants to try should be able to find out the traits in which this expression resides. All he must do is to vary his scrawl systematically. If his first mannikin looks stupid and smug, another with his eyes a little closer to the nose may look less so. By a simple reshuffle of these primitive traits, our lonely hermit will find out how these elements and their combinations affect him and us. Thus a little experimentation with noses or mouths will teach us the elementary symptoms, and from here we can proceed, simply by doodling, to create characters. . . .

Töpffer's method--to "doodle and watch what happens"--has indeed become one of the acknowledged means of extending the language of art.

E. H. Gombrich
Art and Illusion, pp. 339-340,356
(Emphasis added)

I.A.1. THE ROLE OF DIRECT GRAPHICAL INTERACTION IN THE SYNTHESIS OF ANIMATED VISUAL DISPLAYS

This dissertation is a study of ways in which man aided by a computer may synthesize animated visual displays. It assumes that the animator and the computer will be working in the kind of interactive context that is by now well known and whose utility in computer graphics is well appreciated. Work in fields as diverse as the design of large sculptured shapes like airplanes and automobiles,[32] the layout of integrated circuit masks,[33] textile designing and weaving,[34] mathematical graph theory,[35] and the debugging of computer programs,[36] has established the fact that on-line graphical interaction facilitates man-machine communication about still pictures. (*1) This research explores the role of direct graphical interaction in the construction of motion pictures.

On the basis of experience with static pictures, four favorable aspects of the role of interaction in computer graphics may be distinguished:

(1) The availability of immediate visual feedback of results, final or intermediate;

(2) The ability to factor picture construction into stages, and to view the result after each stage;

(3) The ability to designate commands and pictures directly and naturally, anytime during the course of picture construction; and,

(4) The ability to sketch pictures directly into the computer.

Furthermore, we have seen that the computer simulates not only a passive recording agent in its ability to retain images, but an active medium which transforms the very nature of the sketching process. It can help in the imposition of structure onto pictures and in the transformation of simple drawings into more complex ones, sloppy sketches into precise ones.[39]

Analogous statements on the role of direct interaction apply in the domain of the computer graphics of dynamic displays.

The power of immediate visual feedback in animation is striking. The computer calculates, from its representation of a dynamic sequence, the individual frames of the corresponding "movie." Like a video tape recorder, it plays it back for direct evaluation. A small change may be made, the sequence recalculated, and the result viewed again. The cycle of designation of commands and sketching by the animator, followed by calculation and playback by the computer, is repeated until a suitable result is achieved. The time to go once around the feedback loop is reduced to a few seconds or minutes. In most traditional and computer animation environments, the time is a few hours or days. The difference is significant, for now the animator can see and not merely imagine the result of varying the movement and the rhythm of a dynamic display. Thus he will be led to perfect that aspect of animation that is its core: control of the changing spatial

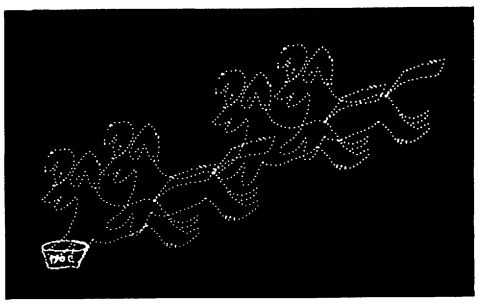
and temporal relationships of graphic information.

Factoring the construction of an animation sequence facilitates the effective use of feedback from early stages to guide work in later stages. Working on individual small subsequences helps overcome the serious practical problems of computer time and space, difficulties documented in Chapter III.B., that could disallow rapid enough calculation and playback. One goal of this research is a conceptual structure that facilitates factorization in time, through the definition of individual frames and of consecutive subsequences, and factorization in space, through the definition of parallel strands of dynamic activity that exist concurrently. Modularity, where possible, is as much a part of good animation practice as it is a part of good programming practice. Several stages in the construction of two animation sequences are depicted and discussed in Figures I.A.1-10.

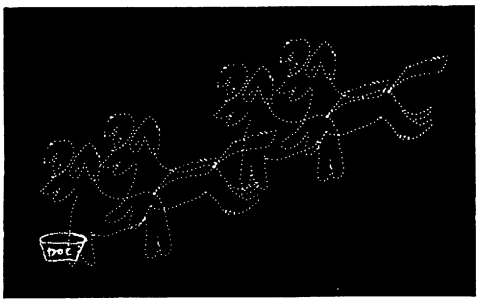
Through direct control over the computer one commands the various aspects of construction and playback. The problems here are somewhat more difficult than they are in the domain of the computer graphics of still pictures. (*2) One goal of the language design of Part II is that the control features of an interactive graphics system be flexible and adaptable. The language allows the user to define his own conventions of interaction--what each action of his is to mean and what the visible response from the computer is to be.

There are at least two distinct roles for direct sketching in the production of dynamic displays. As in traditional animation, a sequence of drawings may be constructed as constituents of individual frames of the movie, as static images existing at single instants of time. If this were all that were possible, then picture change that extends over entire intervals of time could only be synthesized as a succession of individual (temporally) local changes that alter one frame into the other.

The dissertation goes further, for it explains how the computer can be a medium which transforms the very nature of the process of defining picture change, of defining movement and rhythm. Dynamic behavior is abstracted by descriptions of extended picture change. These descriptions may themselves be represented, synthesized, and manipulated through pictures, both static and dynamic. The animator can then define a stream of images with the same facility as he can construct a few simple still or changing pictures, for these pictures both generate and represent that stream of images. Each picture affects all images of the sequence. This means that dynamic control can be exercised globally over the entire sequence. The result is one new conception of what it means to draw an animated film.



(1)



(2)

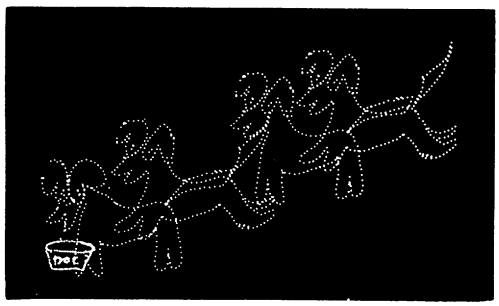
Figures I.A.1-4 are successive stages in the construction of a cartoon depicting a dog dashing to his dinner and then dining. Four frames from each movement are shown superimposed. Further details may be found in R. M. Baecker, Picture-Driven Animation, Proceedings of the 1969 Spring Joint Computer Conference.

(1) A static dog glides towards a bowl. The sketches are by Mrs. Nancy Johnson of Waltham, Massachusetts.

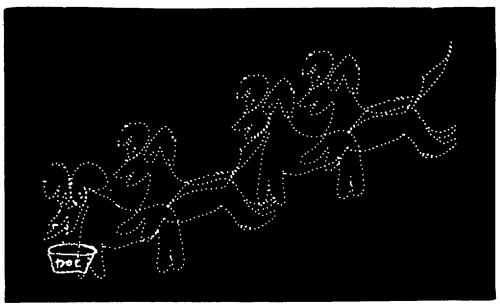
(2) Leg motion has been introduced. Now the dog hops to the bowl.

Figures I.A.1-2

A SHORT CARTOON---STAGES ONE AND TWO



(3)



(4)

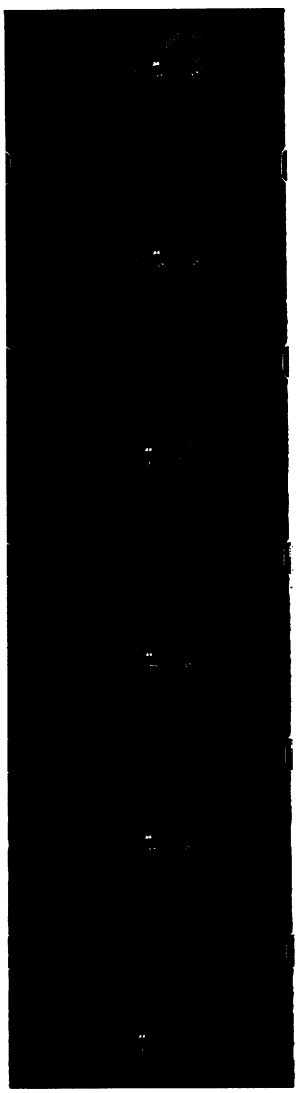
(3) Eager for dinner, the dog wags his tail.

(4) Slurp goes his tongue, lapping up the milk.

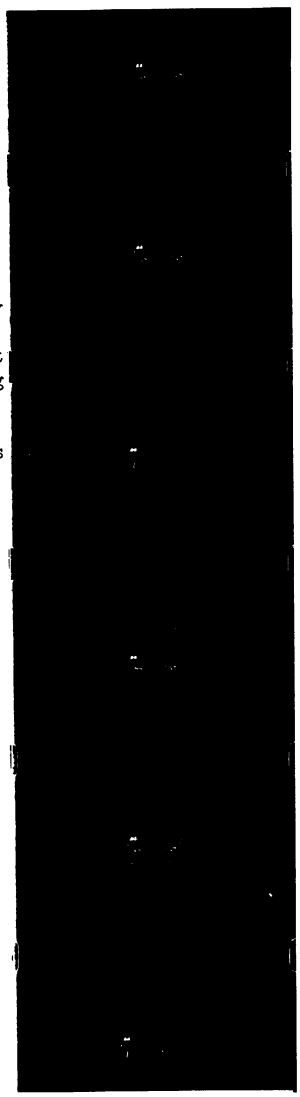
None of these intermediate stages need be transferred to film. Each can be evaluated directly at the interactive animation console, and film used only when the movie is complete.

Figures I.A.3-4

A SHORT CARTOON---STAGES THREE AND FOUR

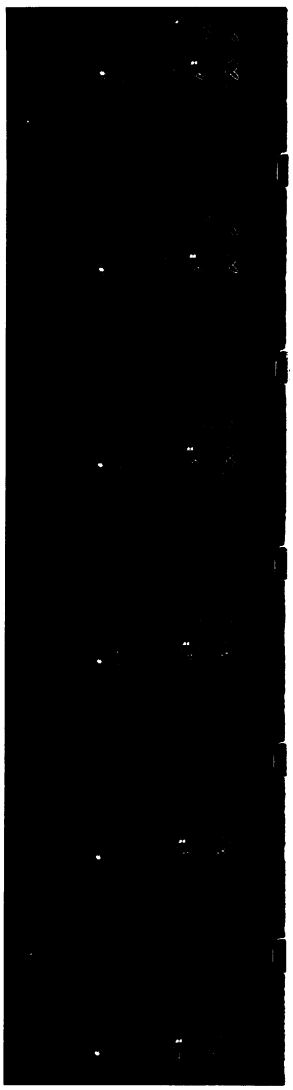


One movie may easily be transformed into another. The result of the first working session was this happy, hopping crocodiless. Frames spaced at uniform intervals in time are shown top to bottom, left to right. The sketches are by Miss Barbara Koppel of Chicago, Illinois.



(5) (6)

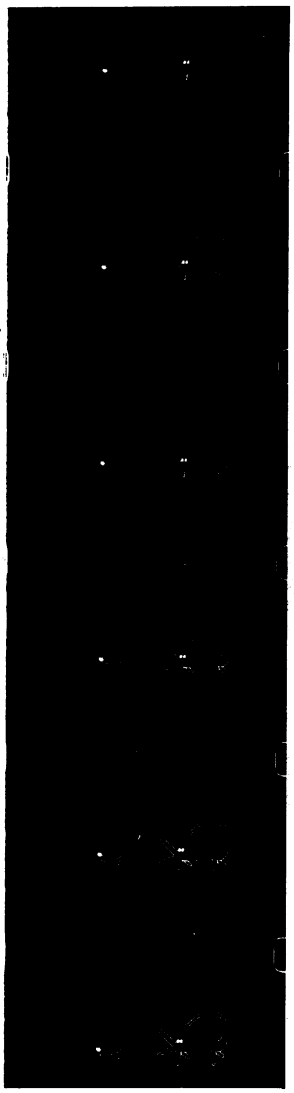
Figures I.A.5-6
A CAVORTING CROCODILESS



(7)

Then we decided to alter the theme of the movie. At the second working session, we shifted spatially the original movement (with one command to the system, a good illustration of global control over an entire sequence). We then added the crocodiless' mate, and caused him to respond as is shown on the following page.

The initial frames of the new movie appear here; the concluding frames appear on the next page.

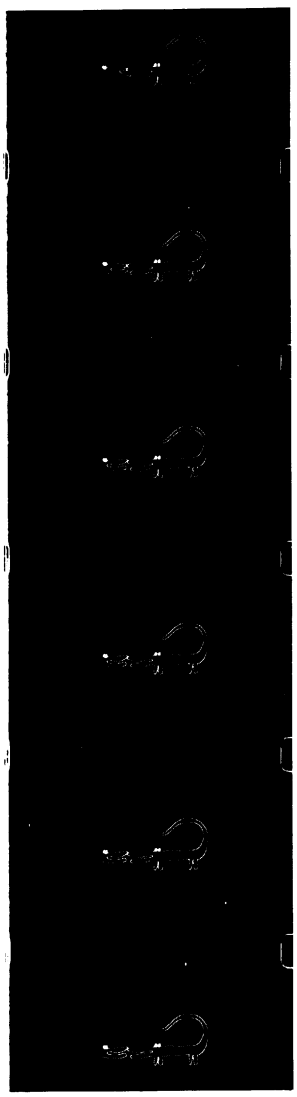


(8)

Figures I.A.7-8

TWO CAVORTING CROCODILES---THE BEGINNING

Interactive computer-mediated animation derives much of its power from the ease and fluidity with which such transformations of one dynamic sequence into another may be executed and evaluated.



(9) (10)

Figures I.A.9-10
TWO CAVORTING CROCODILES---THE END

I.A.2. THE COMPONENTS REQUIRED TO REALIZE AN INTERACTIVE COMPUTER-MEDIATED ANIMATION SYSTEM

Interactive computer-mediated animation is the process of constructing animated visual displays using a system containing, in one form or another, at least the following eight components:

Hardware:

(1) A general-purpose digital computer.

(2) A hierarchy of auxiliary storage. This is listed separately to emphasize the large quantities of storage required for the data structures from which an animation sequence is derived and for the visual images of which it is composed.

(3) An input device such as a light pen, tablet plus stylus,[53-55] or wand,[56] which allows direct drawing to the computer in at least two spatial dimensions. (*3) The operating environment must, upon user demand, provide at least brief intervals during which the sketch may be made in real time. This means that the animator must be able to draw a picture without any interruption from the system. Furthermore, the computer must record the "essential temporal information" from the act of sketching. Sampling the state of the stylus 24 times per second often suffices for our purpose.

(4) An output device, such as a standard computer display scope or a suitably modified TV monitor, which allows the

direct viewing of animated displays in real time. (*4) Playback at 12 or 24 frames per second is usually required. This feature is essential to enable the interactive editing of animation subsequences. The final transmission of a "movie" to the medium of photographic film or video tape can but need not use the same mechanisms.

Software:

(5) A "language" for the construction and manipulation of static pictures.

(6) A "language" for the representation and specification of picture change and the dynamics of picture change. We shall introduce in this work methods of specifying dynamics not possible with traditional animation media and not yet attempted in the brief history of computer animation.

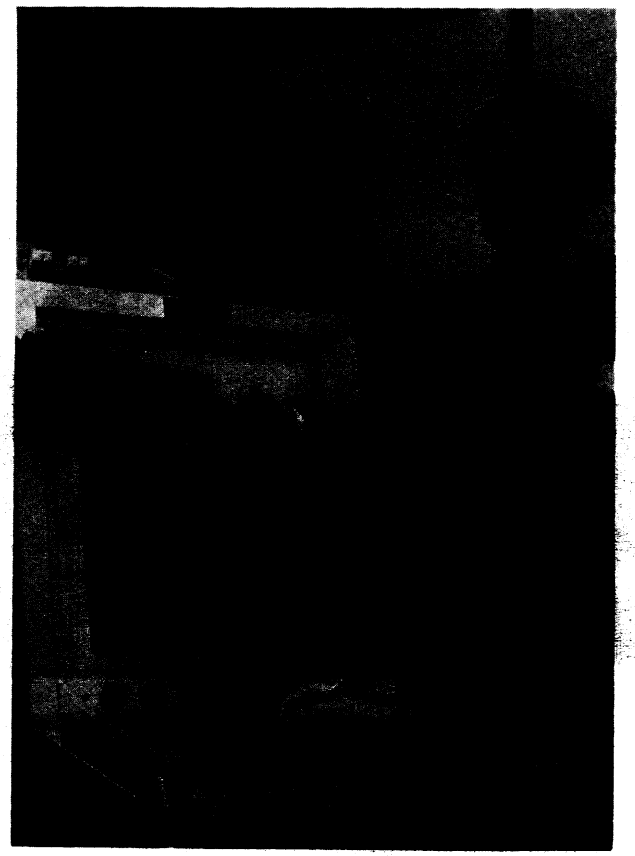
(7) A set of programs that transform the specifications of picture structure and picture dynamics into a sequence of visual images.

(8) A set of programs that stores into and retrieves from auxiliary memory this sequence of visual images, and facilitates both its real time playback for immediate viewing and its transmission to and from permanent recording media.

Figures I.A.11-12 portray and discuss a suitable environment for interactive computer-mediated animation. Figure I.A.13 is a block diagram of such a system.

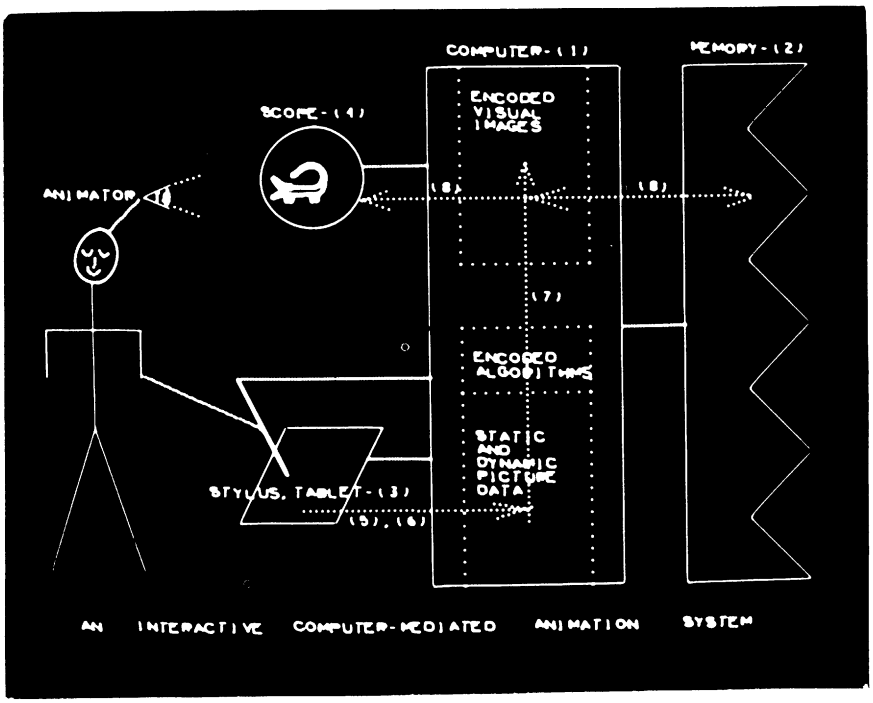
This is a typical interactive computer-mediated animation console. The author is sketching with the stylus on the tablet. There is a CRT for viewing dynamic displays, a storage scope above it, a typewriter, knobs, toggle switches, and a telephone so that the animator may summon help.

Figure I.A.11
AN INTERACTIVE ANIMATION CONSOLE



Interactive computer-mediated animation is often a group activity. Suppose, for example, that an educator, an animator, and a programmer collaborate on an educational film. They can together propose, evaluate, and modify new ideas in a highly fluid manner. They view the visual consequences of each suggestion directly at the console, without the time delay or expense of transferring it to photographic film.

Figure I.A.12
DISCUSSION OF A MOVIE UNDER CONSTRUCTION



Above is a block diagram of a minimal system for interactive computer-mediated animation. The parenthesized numbers refer to the system components defined in the dissertation.

Figure I.A.13

AN INTERACTIVE ANIMATION SYSTEM---A BLOCK DIAGRAM

I.A.3. A SCENARIO ILLUSTRATING THE USE OF AN INTERACTIVE COMPUTER-MEDIATED ANIMATION SYSTEM

To illustrate the process of animation in an interactive computer graphics environment, we present a scenario. The example could be executed with the GENEralized-cel animation SYStem, a picture-driven animation system implemented on the M.I.T. Lincoln Laboratory TX-2 computer. All capabilities purported to GENESYS are operational or could be made so with minor additions. The written form of the interactive dialogue has been adjusted to increase its clarity.

We want to see a dynamic sequence with two characters that are simple abstract figures, a wedge and a block. The wedge is usually triangular, the block rectangular. The wedge bounces in, goes sproing on top of the block and bounces away. The block quivers and quickly recovers. After a slight delay, the wedge returns, this time coming from the opposite direction, and pounces again. Now recovery is more difficult; the block's reverberations die out more slowly. On the third attempt, however, the block enlarges and then devours the wedge.

How we do it:

ANIMATOR(A): *CALL* GENESYS;

GENESYS(G): *HELLO, GENESYS AWAITS YOUR CREATION;*

[GENESYS either types this response, or displays it in an area of the scope designated for messages and instructions from the system to the animator.]

A: *FORMMOVIE* SPROINGBOINGZAP;

[The animator either types the command name '*FORMMOVIE*,' hits a corresponding light-button with the stylus, or writes an abbreviation of the command name to a character recognizer.[58] (*5) He then types a movie name, 'SPROINGBOINGZAP'.]

G: *FRESH;*

[No such movie exists in the animator's directory. Therefore, the command means that work begins on a totally new one. (*6)]

A: *FORMBACKGROUND;*

[A. wants to define a subpicture that will be visible in all frames of the sequence.]

G: *SKETCH IT, MAN;*

A: .....

[A. sketches a background of two scowling clouds and a smiling sun, drawing with the stylus on the tablet. What he draws appears immediately on the display scope. He indicates that he is through by giving a termination signal, such as lifting the pen high above the tablet.]

G: *OK;*

A: *FORMCEL* 1 *in the class* P.WEDGE;

.....

[The wedge, in its various instantaneous sizes and shapes,

is to be represented by a unique set of subpictures, called a cel class. He sketches one version of the wedge as a unique subpicture, or cel, in the cel class P.WEDGE.

.....

He then sketches a square as a unique cel in the class named 'P.BLOCK.' Now the wedge and the block, unmoving, appear on the scope along with the clouds and the sun.]

.....

A: *SKETCHPCURVE* P.WEDGE;

.....

[A. sketches the path of the desired motion of the wedge, mimicking the movement with the action of his stylus. Bounce...bounce...sproing...bouncebouncebounce goes his hand. The act of mimicking a continuous movement is called a p-curve. A visible, growing trail of symbols mirrors the motion of the stylus. The resulting trail is later removed from the display.]

.....

A: *PLAYBACK;*

[Playback the current version of the movie. Bounce... bounce...sproing...bouncebouncebounce goes the rigid wedge across the scope.]

.....

A: *EDITFRAME* 41;

.....

[Assume that the wedge strikes the block in frame 41. Viewing the sequence in slow motion, A. notices that the wedge appears to overlap the block in a way that destroys the illusion of its striking and rebounding. Since only its location in frame 41 in incorrect, he alters the vertical position in that frame by using a knob (shaft-encoder) under the scope.]

.....

A: *PLAYBACK;*

[With that adjustment made, the wedge's first movement is found to be satisfactory.]

.....

A: *FORMCEL 2 in the class* P.BLOCK;

.....

[A. sketches the block in another shape, that is, he defines the second cel in the class P.BLOCK. This is followed by several more shapes and sizes. The images are ones that the animator thinks will be useful in synthesizing the reaction of the block to the blow from the wedge. Each depicts the block somewhat flattened, now more or less rectangular, and with ripples in its top surface that will be combined into a 'boinggg.']

.....

A: *TYPESELECTIONSFROM* P.BLOCK;

.....

[He types in a sequence of integers, each of which designates a choice of one of the drawings of the block. Each succeeding choice selects the cel to be displayed in the next frame. Of course only one state of the block is visible in a frame, when played back.]

.....

A: *PLAYBACK;*

[Now the block is squashed and then quivers in response to the blow from the wedge. The total sequence is

bounce...bounce...sproing...bouncebouncebounce...
.....boinggg.....

.....

Further refinements to the quivering are made. These include the resketching of one cel, and alterations of the sequence and rhythm of transitions among cels. The effect of each modification is immediately viewed by calling *PLAYBACK.*]

.....

A: *REPEATDYNAMICPATTERNOF* P.WEDGE;

[This appends a copy of the dynamics of the wedge's bouncing motion to its existing dynamics, thus causing the movement to be repeated. Since only one cel representing

the wedge has been introduced, its dynamics are generated solely by translational motion.

.....

He similarly causes the reverberation of the block to be repeated. In this case the dynamics are generated solely by sequences of cel selections.]

.....

A: *TAPRHYTHM* BOINGINTERVALS;

.....

[The block's second reaction should consist of several distinct reverberations, each delayed somewhat in time from the last. A. can feel or intuit the rhythm of the desired successive movements better than he can rationalize it. Hence he goes tap...tap.....tap.......tap on a push-button.

.....

Using techniques we shall not detail here, he isolates the pattern of block selections which yields a single reverberation, and repeats it at intervals determined by the tapped rhythm.]

A: *PLAYBACK;*

[Now the action is

```
bounce...bounce...sproing...bouncebouncebounce...
.........................boinggg.............
...bounce...bounce...sproing...bouncebouncebounce.............
.............................boinggg......boinggg.....boinggg.
.......................
.............boinggg...]
```

.....

A: *EDITDYNAMICS;*

.....

[GENESYS produces a display, arranged on a common horizontal axis which represents movie time clocked in frames, of the changes with time of the selection sequences and the X and Y coordinates of the wedge and the block. (*7) There are then available a variety of editing commands for operating upon these graphs and thereby changing the movements they represent. For example, the animator decides that more speed and acceleration in the second hopping movement would better portray the wedge's mischievousness. He also recalls he intended that this motion be from left to right, opposite in direction from the first one. Hence his first command negates the section of the X waveform of the wedge which defines the second movement. He then latches on to this section of the curve with his pen, and compresses it horizontally. This has the effect of speeding up the motion. Finally, he resketches part of the waveform so that there is more acceleration in

one part of the movement. He also edits somewhat the selection sequences of the block, adding some variety in the successive reverberations.]

.....

A: *PLAYBACK;*

.....

[The final task is to construct the wedge's third attack and its unhappy end. New cels for both the wedge and the block are constructed. A variety of techniques may be used, again including the mimicking of new continuous movements and the alteration of existing ones, the definition of new selection sequences, and some fine adjustments on individual critical frames.]

.....

A: *PLAYBACK;*

[Thus, the final sequence is

```
bounce...bounce...sproing...bouncebouncebounce...
.........................boinggg.................
...bounce..bounce..sproinggg..bouncebouncebounce.................
.............................boinggggg......boinggg....boingg...
.........................bounce..bounce..sproin
..........boing.................................Z^gA P !!!]
```

A: *SAVE* SPROINGBOINGZAP;

[The movie is saved under the name 'SPROINGBOINGZAP,' and is available for playback or further refinement at any time.]

G: SPROINGBOINGZAP *IS SAVED; GOOD BYE.*

I.A.4. IMPLICATIONS OF THE SCENARIO

(1) Approximately 200 frames are generated from fewer than 20 cels. These cels may be constructed with very limited tools, specifically, programs that accept direct sketches and that enable selective erasure of picture parts. Nonetheless, great power results from the animator's ability to control and evaluate dynamic combinations of a few static images.

(2) Immediate playback encourages trial-and-error experimentation to achieve desired visual effects.

(3) A variety of static images, analytical graphs of picture action, depict the time dependence of dynamic picture parameters. An example is the waveform representing the wedge's changing horizontal position. Viewing such static representations aids the understanding of existing animation sequences; resketching or editing them changes the actual dynamic behavior accordingly.

(4) The animator may in real time mimic aspects of dynamic behavior. His movement and rhythm are recorded by the system for application in the movie. This occurs when the bouncing of his stylus motion is used to drive the wedge, and when the tapping of a push-button is used to determine the rhythm of the recurring reverberations of the block.

(5) Three aspects of dynamic behavior appear in the example:

path descriptions, or potentially continuous coordinate changes;

selection descriptions, or recurring choices of cels from a cel class; and,

rhythm descriptions, or temporal patterns marking events.

The pictures (3) and actions (4), through which direct control over dynamics is exercised, are representations of these three kinds of global descriptions of dynamics.

(6) Global operations (3)-(4), which alter dynamic behavior over entire intervals of time, may be supplemented where necessary by local operations, which adjust individual frames. An example is the positioning of the wedge with respect to the block at the moment of impact.

FOOTNOTES -- I.A.

(*1) Primarily for the benefit of readers not trained in computer science, here are numerous references to articles on interactive computing and computer graphics: The historical roots are found in the classic papers of Bush[37] and Licklider.[38] I. E. Sutherland's SKETCHPAD, still fundamental to workers in the field, was the first interactive computer graphics system of significant generality.[39] A readable introduction to computers, methods of man-machine communication, and various applications is a recent issue of Scientific American.[40] The most enthusiastic and spirited descriptions of the potential of interactive computing are those by Licklider.[41,42] There are many current survey papers, including one in each Annual Review of Information Science and Technology that reports on and refers to yearly progress.[43-47] Also of considerable interest are a 1966 paper on ten major unsolved problems of computer graphics (most still essentially unsolved, many relevant to achieving economically viable computer animation),[48] an article stressing hardware-software trade-offs in the design of computer graphics systems,[49] and more-or-less detailed descriptions of available display hardware.[50,51] Finally, a recent book surveys the entire field of information display, including photometry, colorimetry, image analysis, optics, recording media, cathode-ray devices, film-based projection systems, light valves, lasers, electroluminescent devices, and laser holography, and lists extensive references to each of these topics.[52] The use in movie-making of technologies other than film and cathode-ray devices should soon be explored.

(*2) Not only the "result" of a user action, that is the sequence of designated commands and data, but also the dynamics of the action, must be properly recorded and interpreted by the system. Furthermore, extended user actions may exist concurrently with extended system actions, such as the dynamic display of a picture.

(*3) References 53 and 54 describe two tablets currently much in use. Reference 55 presents a device called the comparator, which should be added to tablet hardware to simulate the light-pen's role in the direct designation of picture parts by pointing at them. The comparator generates an interrupt whenever the display beam comes close enough to the stylus position.

(*4) Reference 57 describes a novel hardware configuration, potentially useful for animation, in which the display

is projected through the glass of the horizontal tablet. The TX-2 hardware, like most, has a vertical scope and a horizontal tablet mounted beneath it.

(*5) Serious problems can arise in the display of a menu of light-buttons corresponding to available system commands, because the area on the scope allocated to such functions, in fact, the entire area of the scope, is often inadequate for sets of commands large enough to be useful. Sophisticated layout algorithms, and paging and windowing mechanisms may be devised to combat the difficulties. GENESYS currently employs some paging, and relegates many commands to typewriter control.

(*6) A typical alternative response would indicate the length in frames of the existent movie named 'SPROINGBOINGZAP,' and the date and time of the most recent session in which it had been constructed or modified.

(*7) A similar display is depicted in Figure I.B.1, in which the graph of a selection description appears near the top of the picture, and the waveform representing the change with time of a continuous coordinate appears near the bottom of the picture.

I.B.1. THE INDIVIDUAL CONSTRUCTION OF EACH FRAME IN THE SEQUENCE

Animation sequences have traditionally been synthesized through the individual construction of frames, with the illusion of a continuum of time later being attained through rapid playback of the discrete instants of time. This approach is the only one possible when constructing a series of pictures which defy regular or formal description, and which require unique operations on each frame.

Stephenson describes how the burden of drawing 1440 pictures for each minute of film has been lessened in commercial animation, and how this has affected both the product and the process of animation:[3]

> . . .There are various means of economising. By using layers of transparent celluloid, and by painting or drawing different parts of the scene on different layers, backgrounds or characters which are stationary can be used again and again, and only the part actually moving at the time (lips, fingers, eyes etc.) need be re-drawn. Again an arm on a separate cel can be moved by tilting the cel instead of redrawing. A second means of economy possible with slower movements, is to repeat the same drawing for two (even three) frames. This is equivalent to projection at 12 frames a second, intervals are adjusted to correspond, and the eye accepts movement as normal or nearly normal. In more recent cartoons less realistic movement is accepted. As John Halas says "Now we can get away with four drawings a second whereas once twenty-four were necessary."
>
> The third way of lightening the labour of drawing involved in the cartoon is by simplifying the style. The patient composition of the painter in oils or water colour, the intricate traceries of the pen-and-ink artist or the etcher, the careful building up of subtle colour

effects, are outside the range of the cartoonist. Simple line, clear, readily-grasped colour effects, are what the animated artist must go for. The economy of drawing dictated by the conditions of production may seem a handicap. The cartoon will never emulate the wealth of detail which enriches the picture galleries of the world. But when we come to the conditions of viewing, the cartoon's simplicity is an advantage. If the moving cartoon were as complex as the static painting the viewer would never be able to grasp it. A clear, quickly-understandable composition is essential.

... (*1)

A fourth method of meeting the demands of the cartoon medium is by organization and division of labour. Carl Fallberg writes in The American Cinematographer: "Assembly-line methods are essential. It is technically possible for one person to do everything from the first preliminary story sketches to photographing the finished drawings on film, but there is such an infinitude of detail involved that the number of hands doing the work simply must be multiplied."

In every animated film-studio of any size the work is divided up and different people specialize in particular jobs. In a small studio the workers will know each other's jobs, there may be an interchange of work or at any rate a rewarding feeling of group effort and mutual appreciation of each other's skills. In the large studios it may be more like a factory process. There are individual variations in different studios but the general procedure is as follows. Before the animation drawing is started there are several preparatory stages--story-board, work-book, layout, model sheets, soundtrack, charts, dope sheets. The story-board is the same as for any film: a series of small sketches with enough description to enable the plot of the film to be followed. The work-book is a much more detailed, almost photo-by-photo analysis without drawings, describing the action in words and giving dialogue or other sound to accompany each action; it is a kind of film script. Layout and model sheets are preliminary drawings to determine the type and relative size of the characters and the style of the film. The soundtrack charts show the music, bar by bar, related to the visuals, and the dope-sheets or camera exposure charts,

> schedule in more detail each single exposure. The film is then fully planned and the work of execution begins.
>
> The following now set to work: the background artist, the key animator, the in-betweeners, the inker, the painter, the checker, the cameraman, and the film editor. The key-animator draws the key positions of each movement, the in-betweeners copy his work, varying it slightly to provide the necessary movement between one position and another. Then the inker blocks in the outline on the celluloid sheet and the painter or opaquer fills it in with the correct colour. Checkers ensure that the cels are properly lined up and matched and in the correct sequence for the cameraman who photographs the drawings. An animation camera and its rostrum are especially designed to photograph drawings or models or objects, photo-by-photo, frame-by-frame.
>
> One sequence or scene of the film will be completed at a time and viewed. Finally, the various sequences will be assembled by the editor and the completed film is ready for viewing.

It is this division of labor, this dispersal of the creative process which separates the artist from the medium.[4] Another result is the continuing dramatic rise, faster than the GNP, of the cost of animation.[4] In large studio operations, salaries for producers, directors, designers, layout artists, studio managers, and those others named above typically consume half of the cost of a film.[2]

A serious weakness of conventional frame-by-frame animation is that there are no efficient methods of making changes to a movie stored on photographic film or video tape.

On the other hand, film's high resolution and resulting capacity to store and convey information allows diverse kinds of textures and color to be reproduced. Dynamic collages of a variety of cut-out material, puppet animation, and McLaren's

technique of drawing and painting directly on film all achieve radically different tonal qualities. (*2) Such qualities are presumably difficult to preserve in computer processing. However, we need not try to imitate them when we instead can develop new and interesting styles unattainable without a computer, such as those created by Harrison at Computer Image and Vanderbeek using BEFLIX.

Despite the computer's flexibility as a sketching and resketching medium, it also appears wasteful to use a large general-purpose digital computer merely to reproduce conventional frame-by-frame animation techniques. In this dissertation we seek ways to use creatively the great computational capacity of such a machine.

With the aim of streamlining the techniques of cel and cut-out animation, the National Film Board of Canada has undertaken to control the movements of their animation stand with a small digital computer.[59] (*3) It will be interesting to compare and integrate the results of their approach with those of Computer Image and this dissertation, as all may contribute towards the design of a more flexible, more responsive animation-machine.

I.B.2. THE INTERPOLATION OF SEQUENCES OF FRAMES INTERMEDIATE TO PAIRS OF CRITICAL FRAMES

The technique of interpolation has long been used to cut costs and reduce the burden of picture construction which is placed on the key animator. Interpolation occurs when the key animator asks his assistants to fill in the frames intermediate to a pair of critical instances of transition. In a typical cartoon, for example, the key animator may himself sketch only the following: gasp of horror, arm cocked for ferocious swing, arm thrusting forward as target ducks, arm stretched as if made of rubber, and arm zooming off into outer space.

It has been suggested that part of the interpolation process could be mechanized, and we have described in the Introduction the only published attempt to do this.[28] We shall discuss interpolation in its general formulation no further. Limited instances of interpolating sections of movement, however, will reappear, particularly in I.C.

I.B.3. THE GENERATION OF FRAMES FROM AN ALGORITHMIC DESCRIPTION OF THE SEQUENCE

The generation of a sequence of frames from a formal algorithmic description is a process characterized by:

(1) the need to use a computer, for it is the only animation medium which can follow and execute with ease a complex algorithm;

(2) generality, that is, applicability to a large class of regularly-structured pictures;

(3) representational power, or the compactness with which interesting animated displays may be formulated; and,

(4) flexibility and adaptability, or the ease with which a variety of alterations may be made to a movie expressed as an algorithm.

One primary source of representational power and flexibility is the algorithm's temporally global quality--it specifies the picture state over an entire interval of time, and not merely at individual frames.

We have noted in the Introduction that, since formal picture descriptions are usually expressed as written programs in a language such as BEFLIX,[5,6] or as sequences of directives in a typewriter-controlled command language such as CAFE,[26] artists have found it difficult to adopt the computer as a new medium.

The process of picture-driven animation attempts to bridge the gap between traditional and computer techniques by exploiting direct graphical interaction and the descriptive capability of pictures and sketches. As we shall now see, the animator creates a movie by combining algorithmic definitions, drawings, and representations of movement and rhythm.

I.B.4. PICTURE-DRIVEN ANIMATION

Picture-driven animation is a new process that augments harmoniously the animator's traditional techniques, that reflects and extends the ways of thinking to which he is accustomed. Within his intuitive "language" of pictures and sketching and mimicking, he may synthesize both components of frames, called cels, and generative descriptions of extended picture change, called global descriptions of dynamics.

Global dynamic descriptions are data sequences, whose successive elements determine critical parameters in successive frames of the movie. Algorithms embedded in a picture-driven animation system combine cels and dynamic descriptions to produce visible picture change. The animator defines and refines pictorial representations of dynamic descriptions. These data sequences then "drive" the algorithms to generate an animated display. Hence the process is called picture-driven animation.

The process is powerful because it is easy to achieve rich variations in dynamic behavior by altering the data sequences while holding constant a few simple controlling algorithms. The data sequences precisely determine the evolution of recurring picture change, within the constraints set by a choice of controlling algorithms.

The remainder of I.B. introduces the three kinds of global dynamic descriptions and some useful algorithms for

which they may be driving functions. I.C. describes and analyzes the art of defining and refining dynamic descriptions. We adopt the following classification:

A <u>global dynamic description</u> is either

a <u>movement description</u>, which is either

a continuous movement description = a <u>path description</u>, or

a discrete movement description = a <u>selection description</u>; or,

a <u>rhythm description</u>.

I claim that the terminology is appropriate, that these descriptions are conceptually meaningful abstractions of movement and rhythm, a kind of "vocabulary for dynamics," a vocabulary for expressing picture change.

I.B.5. THE RELATIONSHIP, IN GENESYS, OF CELS AND CEL CLASSES TO GLOBAL DYNAMIC DESCRIPTIONS

Since the following discussion relies heavily on illustrations of the use of GENESYS, we now describe how it combines cels and global dynamic descriptions to produce dynamic displays.

GENESYS permits users to impose dynamic picture behavior independently of the definition of static picture structure. Static pictures are generated by the traditional techniques of sketching and individual construction, and may then be used in one of three ways:

(1) A static picture may serve as the constant background for an entire animation sequence, as did the scowling cloud and smiling sun in 'SPROINGBOINGZAP.'

(2) A static picture may be introduced in a single unique frame of the final animation sequence. Frames totally generated in this manner have therefore been defined by the approach of individual construction.

(3) A static picture may be defined to be a unique cel. Cels are then grouped into cel classes. The term 'cel' is taken from conventional animation practice. We have seen that it means a sheet of transparent celluloid on which is painted some separable aspects of the picture scene, part of the

background (group of trees, or clouds, or a rocket-ship) or even the foreground (hat which is to fly off, garbage can which is to be kicked or sprinkled). Examples of two cel classes in GENESYS are the wedge and the block of 'SPROINGBOINGZAP', each represented in various sizes and shapes. Other examples are a set of choices for the mouth of a cartoon character, and a set of striped patterns used to create the illusion of a revolving barber pole.

Using the current system, the GENESYS animator sketches or otherwise defines two path descriptions and one selection description for each cel class. The system applies a simple algorithm which in each frame selects and positions one cel from each class according to the data in the descriptions.

Beginning with this very special case, we shall develop and broaden the concept of global dynamic description, showing how the sequences may drive a much wider variety of algorithms, and thereby generate a much more interesting repertoire of dynamic displays.

I.B.6. PATH DESCRIPTIONS

Consider those alterations of static pictures that consist of modifications of continuously variable parameters, such as location, size, and intensity. Their instantaneous values determine the picture's appearance at a given moment. Thus the static picture may be animated by specifying the temporal behavior of such parameters. A representation of the temporal behavior of a continuously variable parameter is called a path description.

The movement of a fixed-geometry picture (cel) in GENESYS is described as the change of two coordinates with time, and is represented by a pair of path descriptions. Their specification may be used to synthesize the drifting of a cloud, the zooming of a flying saucer, the bouncing of a ball, or the positioning of a pointer.

Since the behavioral descriptions of the parameters apply to entire intervals of time, the animation is liberated from a strictly frame-by-frame synthesis. The computer is a medium through which one can bypass the static or temporally local and work directly on the dynamic or temporally global. Movement is represented as it is perceived, as (potentially) continuous flow, rather than as a series of intermediate states.

Chapter I.C. describes how path descriptions may be defined by algorithm or by direct on-line construction.

Free hand sketching is a useful technique when one knows the general shape and quality of a motion rather than an analytical expression for a function that determines it. Modifications of the sketches are frequently invoked after one views the current animation sequence and determines how it is inadequate.

There are two related kinds of pictorial representations of all movement descriptions, static and dynamic.

A static representation of a single path description is a waveform, in which time is identified with a spatial dimension, in common practice the horizontal. The continuous curve at the bottom of figure I.B.1. is a waveform depicting a picture parameter which increases and decreases gradually, then quickly oscillates between large and small. The superposition on a common time axis of several path descriptions, such as plots of the positions of fleeing and overtaking flying saucers, facilitates the refinement of their relative dynamics. In this case it provides the animator with a very powerful, temporally global control over the "feeling" of the chase scene.

A dynamic representation of a path description is an animated display.(*4) The dynamic construction of a path description is achieved by timing the stylus's movement and recording its position at short, uniform intervals such as every 24th of a second. A tangible representation of the path is the display of a sequence of symbols spaced equally

in time. Thus the bouncing of the wedge in 'SPROINGBOINGZAP' may be synthesized by "bouncing" the stylus along some path on the tablet surface, that is by mimicking the desired dynamic behavior.

A path description, in summary, defines dynamic activity that consists of potentially continuous and arbitrarily fine alterations of value. The reader should not be misled by the choice of the word "path." What is meant is a path, or sequence of values, through an arbitrary "continuous undimensional space," through a mathematical continuum. One application or interpretation of this path is the representation of a movement through the location-space of an object, such as a figure's trajectory along the floor of a room. To specify this trajectory, we need two paths, x(t) and y(t), that define the horizontal and vertical position. This application, although the only one operational in the current version of GENESYS, is not the only possible one. Depending upon the picture description capability of the system in which it is used, and the algorithm which it drives, a path description may determine changing locations, intensities, thicknesses, densities, or texture gradients. For example, a pulsating heart could be animated by varying either the size or the intensity of a single heart shape. I.B.10. further elaborates on this point.

A pair of path descriptions may be used to represent the layout of a complex information display, that is, an

ordered sequence of locations at which components of the display are positioned. Here the descriptions convey the movement of an implicit scan of a static picture; they need not necessarily correspond to an actual trajectory dynamically traced in an animation sequence.

There is no requirement that the defined movement be applied always to the same cel, or to only one cel. So that the drifting moon may smile in anticipation of a visit from Apollo-10, changing facial components must be superimposed; the facial expressions must be driven by the same path descriptions as move the moon. The GENESYS animator can accomplish this by making copies of the path descriptions. A more powerful and useful mechanism would allow the binding of the motion of one cel class to that of another cel class, so that the controlling movement is always automatically copied into, or added to, that of the subordinate movement. (*5) This technique, whether executed by the animator or automatically by the system, is often used to constrain various cel classes to move as a unit. The cels of a crocodiless, members of the classes 'body', 'legs', 'jaws', and 'tail', are drawn so that the figure coheres as a static image. If the same paths are then applied to all four classes, it will not disintegrate while moving.

I.B.7. SELECTION DESCRIPTIONS (1)

Consider the algorithm that selects an element of the current frame from among members of a cel class. A good example arises in the synthesis of different facial expressions through the abstraction of discrete shapes and positions of mouth, nose, eyeballs, and eyebrows. One cel class could consist of the two members "eyebrows raised" and "eyebrows lowered." An animation sequence in GENESYS may be achieved by a temporal concatenation of selections from a cel class. A changing facial expression may be achieved by the parallel application of several such sequences of selections, one corresponding to each facial component. In 'SPROINGBOINGZAP', this technique was used to generate the reaction of the block to the wedge. (*6)

A representation of the dynamic selection from a finite set of alternative pictures is an example of the second type of global dynamic description and is called a selection description. It is suggestive to think of a selection description as a single melodic line, the notes of which are selected from a conventional discrete scale. The synthesis of selection descriptions is also aided by the use of pictorial representations, such as a graph of a sequence of steps, where the length of each step denotes an integer number of frames, and its height is a transition from one position to another on the

discrete scale. An example of such a picture appears at the top of figure I.B.1.; the selection description it represents is one which chooses among four alternatives. Superposition on a common time axis of pictures of several descriptions facilitates coordinating the counterpoint of the parallel selection strands.

The use of the term "selection" implies that a mechanism chooses from among a designated set of alternatives. In the example of the animation of a facial expression, the alternatives are cels, images to be introduced as components of frames in a dynamic sequence. A natural question comes to mind--is it meaningful and useful in computer animation to consider selections from among other kinds of entities, for example, algorithms or numbers? The answer, as we shall see in I.B.9., is 'yes'.

I.B.8. RHYTHM DESCRIPTIONS

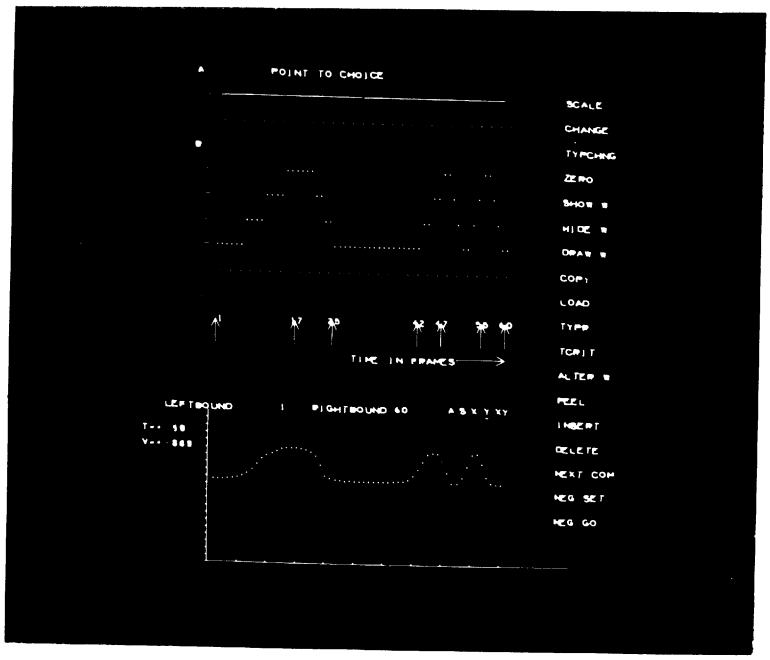
Rhythm descriptions consist of sequences of instants of display time (frames), or intervals between frames. They define patterns of triggering or pacing recurring events or extended picture change. In this context it is suggestive to think of a rhythm description as a pulse train. Each pulse may be used to trigger the same action, or, as we shall soon see, it may trigger one of several activities under the control of a selection description. Examples of the apparent triggering effect of a rhythm occur when we advance the hand of the clock every second, cause a dancing figure to hop at every pulse, or produce a tremor in a house whenever smoke emerges from a nearby factory. In 'SPROINGBOINGZAP' we generated new reverberations in the block at intervals corresponding to a tapped rhythm.

Rhythm descriptions facilitate the achievement of coordination and synchrony among parallel strands of dynamic activity. In this context it is suggestive to think of a rhythm description as a sequence of event markers. The rhythm may be defined with respect to the actions of one cel class, and then used to guide the construction of another action. For example, the animator records the instant when the wedge in its third movement would hit the block, and then adjusts the devouring action of the block to anticipate the arrival of the wedge. He determines the set of frames in which the lead bird of a fleet accelerates, and uses this information

to generate a kind of coordinated action--each bird flies on a separate path, but all accelerate and decelerate simultaneously.

A rhythm description does not by itself define picture change; it defines a beat, a sequence of cues with respect to which picture change is temporally organized and reorganized. Animators have sometimes used metronomes as generators of rhythm descriptions.[2] Proper synchronization of a sound track to the visual part of a film is most critical to its success.[2] In practice, much effort is expended, sometimes aided by special devices, to extract rhythm descriptions from a sound track in such a form that they can be used in animating, or in editing existing animation sequences. (*7)

Hence, rhythm descriptions marking critical instants of time play a key role in the synthesis and editing of movement descriptions. For these operations a rhythm description requires pictorial representation. In Figure I.B.1. it is depicted both as a static pulse train and as a sequence of event markers along the axis of movie time. A direct and simple dynamic input, as we have seen in 'SPROINGBOINGZAP', consists of tapping out the rhythm on a push-button.



The continuous curve at the bottom is a waveform representing a path description. The discrete curve at the top represents a selection description that chooses among four values. The pulse train between them represents a rhythm description. The three pictures are plotted on a common horizontal axis--movie time measured in frames. Picture change occurring in any frame is therefore depicted along a single vertical line. The visible character strings are "light-buttons" which activate GENESYS commands, and messages to the animator.

Figure I.B.1
GLOBAL DYNAMIC DESCRIPTIONS

I.B.9. SELECTION DESCRIPTIONS (2)
--- DYNAMIC HIERARCHIES

With selection descriptions, as we have seen in I.B.7., the GENESYS animator chooses subpictures, or cels, from a cel class. A more general view regards a selection description as a sequence of selectors, functions which choose from a designated and finite yet potentially denumerable set of alternatives. Depending upon the picture description capability of the system in which it is used, and the algorithm which it drives, a selection description may also choose among alternatives that are numbers, picture-generating algorithms, other global dynamic descriptions, or pictorial events or activities.

The dynamic selection from alternative numbers would occur in a system whose pictures could be displayed at one of eight different intensities, or whose lines could be drawn either solid, dotted, or dashed. The dynamic selection among alternative picture-generating algorithms would occur in a system with discrete texture choices, where there is one algorithm capable of filling an arbitrary region with that texture.

A pictorial event is an instantaneous state of an animation sequence, a single frame or one of its constituent subpictures. The subpicture may itself consist of several cels. An example is the initial instant of collision between the wedge and the block in 'SPROINGBOINGZAP'. A pictorial activity

is an extended interval of an animation sequence, or an extended interval of a subsequence contained within it. An example is the 'bounce,bounce,sproing,bouncebouncebounce' of the wedge. A pictorial activity is sometimes called a strand of dynamic activity to emphasize that there may exist other concurrent activities, other parallel dynamic strands.

In a system in which selection descriptions could choose from sets of global dynamic descriptions, pictorial events, or pictorial activities, useful dynamic hierarchies could be established. Suppose, for example, that the animator of a crocodiless develops sequences of selections from the cel class 'jaws' that are visual representations of laughter, smugness, frowning, and crying. When he later wants the crocodiless to laugh, he refers to this pattern of selections as a unit and introduces it as part of a new sequence. Suppose further that a hop, a skip, and a jump of the crocodiless are synthesized. This is done by defining and refining two path descriptions and one selection description for each of the constituent cel classes, 'jaws', 'body', 'legs', and 'tail'. If he then wishes to experiment with varying dynamic patterns of hop, skip, and jump, he defines a selection description which chooses among these three alternative pictorial activities, the sets of dynamic descriptions that form a hop, a skip, and a jump.

I.B.10. GENESYS GENERALIZED

We have seen how a single cel class, a pair of path descriptions, and a selection description are used in GENESYS to generate a dynamic sequence. The argument has also described the far greater role that dynamic descriptions could play, if used as driving functions for a wider class of algorithms. This concluding section suggests how GENESYS, within its design philosophy, can gracefully be augmented to enable dynamic descriptions to play this greater role.

Generalization 1--Projective Transformations of Cels:

Currently in GENESYS, at most one member of a cel class is visible in a given frame. If the selection description assumes the value zero at that frame, no cel is visible; if its value is k, the kth cel is visible. The position of the cel is determined by a pair of path descriptions. Continuous rotation, a very important feature, would be possible if the orientation of the cel could be controlled by a third path description. There would then be as many degrees of freedom as now exist in animating by selecting and arranging paper cutouts and by overlaying drawings done on celluloid sheets.

A straightforward extension of GENESYS would allow the animator to associate with each cel class a sequence of two-dimensional projective transformations, including not only translations and rotations but also scale changes and horizontal and vertical expansions and contractions. Parameters

of the transformation, for instance, a rotation angle and a factor of expansion, would be assigned either a constant value or a sequence of values, that is, a path description. It would then be as easy to depict a rotating space ship fading into the distance as it is to show a space ship of fixed size and orientation zooming across the sky.

Generalization 2--Dynamic Hierarchies:

I.B.9. has introduced the concept of hierarchies of structured dynamic behavior. Combining collections of dynamic descriptions for use as a unit is an economical way of achieving a higher level of discourse. GENESYS should be augmented to include such a macro-capability. The user would then define an event or an activity by listing: the cel class (or group of classes) whose movement is to be saved as a unit; the attributes of the movement to be included, which may be any or all of the x path, the y path, and the selection description; and, the frame of the event or the interval of frames of the activity. (*8) If arbitrary dynamic hierarchies are to be established, then the system must also allow combining defined events and activities into higher-level activities.

What is desired is illustrated by the following hypothetical sequence of commands: *CALL* JAWS BODY LEGS TAIL *the* CROCODILE; *CALL* X Y S *of* CROCODILE *fromframe* 25 *throughframe* 48 *a* HOP; *APPEND* HOP *tothemovementof* CROCODILE; *APPEND* HOP

tothemovementof CROCODILESS. The first command groups a set of existing cel classes into a unit called a 'crocodile'. The second command extracts the fragments of the twelve dynamic descriptions that make the crocodile hop. The third command appends a copy of the hopping motion to the crocodile's existing movement. The fourth command, which assumes that a crocodiless has been similarly formed from four cel classes, causes the crocodiless to hop in the same way as did the crocodile.

Generalization 3--The Generalized-Cel:

A generalized-cel is, roughly speaking, a picture so defined that its appearance in a given frame of the dynamic display is determined by the values of a set of associated movement descriptions. (*9) Cel classes as they currently exist in GENESYS, and cel classes under arbitrary projective transformations, are special kinds of generalized-cels.

As an example of a generalized-cel, consider the following picture--a textured five-pointed star, whose location in two dimensions, orientation, and density of texture are continuously variable, and whose edges are displayed either solid or dashed and in one of eight different intensities. Suppose that these four continuous and two discrete parameters may be varied throughout an interval of a dynamic sequence. Then an instance of the star appearing in a particular frame depends upon the values of four path descriptions and two selection descriptions.

The GENESYS user should be able to commission a generalized-cel in the sense that he instructs a programmer to embed into his version of the system a special algorithm, one which takes a number of movement descriptions as parameters and from their values in each frame computes a picture. This algorithm will be expressed in the language in which GENESYS is implemented, for one cannot write programs in the command language of GENESYS. GENESYS's design, on the other hand, should allow the new command to be integrated smoothly into the system. Specifically, it must guarantee that all tools available for the editing of X, Y, and selection descriptions are available for the editing of the new movement descriptions.

Implementing Generalizations 1 and 3 would augment the class of dynamic images that can be produced with GENESYS; implementing Generalization 2 would only facilitate the production of certain sequences.

In conclusion, we have seen that GENESYS is a picture-driven animation system in which a few simple controlling algorithms combine cels and dynamic descriptions to form animation sequences. The algorithms are fixed and embedded in the interpretation of GENESYS's command set. At some point the animator may discover that the set of available

algorithms restricts the class of dynamic displays that he can generate and modify with ease. Already in this section, we have proposed a concrete schemata for additions to the system. The animator's only current recourse is to persuade the programmer maintaining GENESYS to modify or augment the system. These additions must be made in the language with which GENESYS is implemented, a language unrelated to the command language used by the GENESYS animator. For eventual users of APPL, the system described in Part II, there will be a nicer solution. The user will be given the capacity for picture-driven animation, and also the ability algorithmically to define picture change and algorithmically to control the interpretation of the sketches and pictures through which he communicates with the computer. The algorithms may be expressed within the language of the system, using the command set of APPL. This means that APPL will be an open-ended, multi-purpose animation system.

On the other hand, there are advantages to holding fixed a few simple algorithms, and separating the imposition of dynamic behavior from the definition of static picture structure. We have discovered that even these simple algorithms, when augmented with powerful tools for manipulating movement and rhythm, yield a useful animation system. Furthermore, I would estimate that this system, GENESYS, could be reimplemented on the TX-2 in 4-6 man-months, whereas APPL

might take 1 to 2 man-years. Finally, the process of picture-driven animation is more transparent in a system like GENESYS than in one like APPL, thus making it easier to analyse and clarify the art of defining and refining global dynamic descriptions. That analysis is the task of the next chapter.

FOOTNOTES -- I.B.

(*1) Stephenson,[3] pp. 16-19. At this point, he also notes:

> "As a further compensation the added dimension of movement gives the cartoon an advantage which static art has been reaching out for over the centuries. Even the simplest drawing can be made interesting when things start moving. The composition of the cartoon lies not only in the arrangement of lines and masses in each individual picture, but in the relationship of one picture with the next--contrast or harmony will exist between picture and picture, as well as within the individual image. Thus the cartoon, though lacking some of the possibilities of painting, nevertheless has a richness of its own which offers infinite possibilities and presents a serious artistic challenge to the film-maker."

(*2) See Footnote 1, Introduction.

(*3) The term "animation stand" refers to the animation camera and the rostrum upon which cels and cut-outs are arranged and then photographed.

(*4) It need not necessarily be an animated display of the object that is ultimately being driven by the path description.

(*5) Implementation of this mechanism is straightforward provided one disallows any circularity in the relationships between movements. In other words, if b's movement depends on a's, and c's on b's, then a's cannot be defined in terms of either b's or c's. Allowing circular relations leads to the problem labeled by I. E. Sutherland as "constraint satisfaction."[39] See also I.B.10. and Footnote 8 (Ch.I.B.), and I.E.2.

(*6) There is an interesting difference between the cel class {"eyebrows raised", "eyebrows lowered"} and the cel class used to generate the reaction of the block. In the former example, each cel is a discrete identifiable visual state, which as a static image has significance in terms of its symbolic relationship to other members of the class. The latter case, on the other hand, illustrates that the cels need not represent static states to which we can attach nice verbal labels.

They are in a sense visual artifacts required for the synthesis of the dynamic from the successive presentation of the static.

(*7) Francis Chagrin, a highly experienced composer in animated films, writes in Halas and Manvell,[2] p. 238:

> ". . .more often than not, the music is actually recorded first, then measured carefully (or charted) and all the exact points of emphasis marked and handed over to the animators. . . sound and vision have equal importance; they are equal partners, . . . preparing for it scrupulously by synchronizing their moods and their movements. . . both telling the same story at the same time. . . in perfect harmony and with single-mindedness. I do not mean in unison; the harmony may be brought about by parallel movement or by counterpoint."

(*8) The ability to group cel classes will also facilitate constraining these groups to move as a unit.

(*9) The concept of a generalized-cel parallels what Max Bense calls an "aesthetic object." See Reichardt,[25] p. 72.

I.C. THE ART OF DEFINING AND REFINING GLOBAL DYNAMIC DESCRIPTIONS BY USE OF PICTORIAL REPRESENTATIONS

I.C.1. SPECIFYING GLOBAL DYNAMIC DESCRIPTIONS

Six general approaches to the specification of dynamic descriptions may be distinguished:

(1) The sketching or mimicking of a new pictorial representation of the description.

Examples: Mimicking the 'bounce,bounce,sproing,bounce' of the wedge in 'SPROINGBOINGZAP'; Drawing a rapidly oscillating waveform to represent the path description of the intensity of a 'burning sun'.

(2) The editing or refinement of an existing pictorial representation of the description.

Examples: Editing a picture of a selection description to lengthen one reverberation of the block in 'SPROINGBOINGZAP'; Resketching part of a waveform to add acceleration to the horizontal movement of the wedge.

(3) The direct algorithmic specification of the data sequence.

Example: Setting $x(t) = x_0 + v_{x0}t$, $y(t) = y_0 + v_{y0}t + (1/2)gt^2$ to determine the trajectory of an object falling to the earth.

(4) The indirect algorithmic specification in terms of combinations of existing data sequences.

Examples: Setting $x_2(t) = x_1(t)$, $y_2(t) = y_1(t) + 1$, so that objects #2 and #1 always move together, #2 remaining 1 spatial unit above #1; Setting $x_2(t) = x_1(t-1)$, $y_2(t) = y_1(t-1)$, $s_2(t) = s_1(t-1)$, so that object #2 follows object #1 along a

trajectory, remaining exactly 1 temporal unit behind #1. $s_1(t)$ and $s_2(t)$ are selection descriptions operating upon the cel classes representing the two objects.

(5) An indirect algorithmic extraction of the data sequence as the succession of values assumed by some aspect, or attribute, of a constituent picture in an existing animation sequence. (*1)

Example: Computing f(t) = Distance between PIC.1 and PIC.2, when some other movement is to depend upon the relative motion of these two subpictures, whose absolute motion is already determined.

(6) A coupling to a real physical process in the external world, such that it transmits a data sequence as (analog) input to the computer.

Examples: Interesting couplings may be to particle collisions, the atmospheric pressure or wind velocity, a music waveform, or, in the case of (1) and (2), a real live animator.

GENESYS contains techniques implementing the first, second, and fourth approaches only. APPL allows in principle arbitrary algorithmic specification of data sequences, as well as the dynamic computation (extraction) of a property and its recording as a movement description.

The ability to accept continuous and discrete time series and sequences of pulses and waiting times is critical if one wants aspects of a movie to be precisely determined by phenomena of the real world, such as the output of scientific

experiments. These direct couplings to physical processes depend upon the existence of appropriate hardware, for example, analog-to-digital converters and input data channels. In the implementation of GENESYS and in the design of APPL we have made no explicit provision for such inputs, since only elementary software is required.

A system embodying all six general approaches, each represented by a wide variety of specific techniques, would allow the truly plastic representation and manipulation of dynamic information. (*2)

I.C.2. WAVEFORMS, P-CURVES, AND OTHER PICTORIAL REPRESENTATIONS OF PATH DESCRIPTIONS

Waveforms and p-curves are two kinds of pictorial representations of path descriptions. Both may be introduced with a single example.

Consider the motion of a figure that goes from one corner of a square room to the diagonally opposite corner by walking along two adjacent walls. We shall ignore the vertical movement and consider only motion of the center of the body in the two dimensions of the plane of the ground. He first walks in the direction of increasing X coordinate, then in the direction of increasing Y coordinate. We further assume that he begins from a standstill, accelerates and then decelerates to the first corner, pauses there for a brief interval while he turns in place, and finally accelerates and decelerates to its destination.

One complete description of this planar movement consists of the functions of the X and Y coordinates versus time. These are depicted in Figures I.C.1. and I.C.2. Such representations of changing picture parameters are called waveforms. Time is depicted, in the waveform, along one spatial dimension. The waveform's construction requires movement of the stylus along that dimension; the display records and makes tangible this movement.

Alternatively, both spatial coordinates could denote the two spatial coordinates of the movement. A natural

correspondence is established between the X(Y) coordinate of the floor and X(Y) coordinate of the medium of the representation (paper, scope face, etc.). Figure I.C.3. depicts such a parametric curve representation of the movement. It illustrates with clarity the figure's path on the floor.

Yet the dynamics of the motion are hidden because the temporal dimension is only an implicit coordinate. This rectified in Figure I.C.4. A stream of symbols is used instead of a continuous trail to depict the path. Characters are spaced along the path at short, uniform intervals of time, such as every 24th of a second. Dynamics are apparent in the local density of symbols. Observe in particular how they cluster where the figure pauses.

The dynamic construction of a path description is a user-driven animated display in which the timing of the stylus's movement is preserved by recording its position in every frame. A tangible representation of the stylus's movement is the dynamic display of a lengthening sequence of characters spaced equally in time. We shall call a parametric curve dynamically sketched in real time a p-curve. The p-curve corresponding to Figures I.C.1.-4. is depicted in Figure I.C.6. We have attempted to convey in a single static image that the p-curve is a dynamic display. Each 2-dimensional p-curve determines two path descriptions. Thus the bouncing of the wedge in 'SPROINGBOINGZAP' may be synthesized by "bouncing" with the

stylus along some path on the tablet surface, that is by mimicking the desired dynamic.

The waveform and the p-curve illustrate two aspects of the representation of time that apply to all pictures of dynamic descriptions:

(1) Pictures and actions are termed _static_ when _real time_ and _movie time_ are independent, when the passage of real time in the display or construction of a dynamic description bears no fixed or useful relationship to the dynamics of the animation sequence eventually driven by the description. Alternatively, pictures and actions are termed _dynamic_ when real time and movie time are closely coupled, when picture change and interactive input are clocked, controlled and coordinated by the computer so that they depict and define the dynamics of the eventual movie. We say that these dynamic processes occur _in real time_.

(2) In the visible record of a dynamic description, movie time can be represented as an explicit axis such as the horizontal; the user then draws movie time explicitly along that axis. Alternatively, movie time can be represented implicitly along a curve by a technique such as the controlled spacing of a trail of symbols.

Table 1 summarizes how these distinctions apply to our definitions of waveforms and p-curves:

	Actions and pictures are dynamic, real-time	Actions and pictures are static, not real-time
Movie time is an explicit axis	XXX (*3)	Waveform
Movie time is not an explicit axis	P-curve Strobe	Point-by-point p-curve (parametric curve)

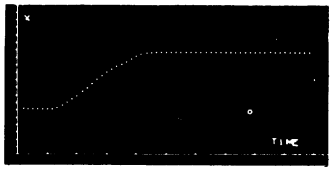
Table 1

PICTORIAL REPRESENTATIONS OF PATH DESCRIPTIONS

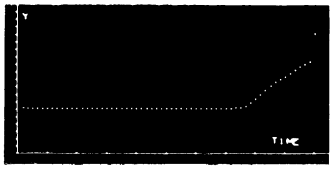
There is a significant qualitative difference between "playing back" the animated display that is a p-curve very rapidly and very slowly. At 12 or 24 frames per second, continuous movements of the stylus are accepted; discontinuities, that is, large gaps between deposited symbols, can be introduced by lifting the pen, thus stopping playback, and then repositioning it again. At 5 seconds per frame, lifting the pen allows each point of the parametric curve to be individually positioned. Since the dynamics of lifting the pen have no effect on the dynamics of the resulting path descriptions, the action of a point-by-point p-curve does not occur in real time.

Still another mechanism for displaying existing two-dimensional movements is called the strobe. (*4) Consider

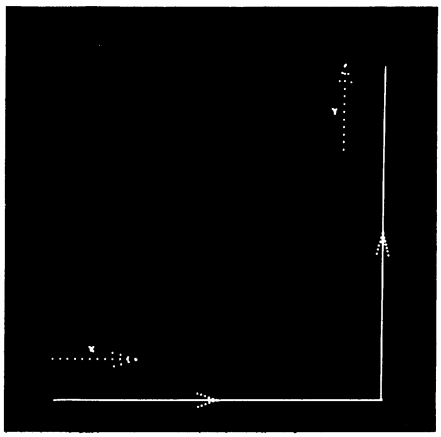
the spatial and temporal superposition of a p-curve and its corresponding parametric curve. Assume that each is represented by visually distinguishable symbols. The p-curve component traces the movement dynamically. The static component allows the viewer to see the entire trajectory throughout the duration of the dynamic trace. Because of the overlapping static component, only the leading point of the p-curve is of interest. Therefore, we replace the growing trail of symbols that is the p-curve with a moving marker that represents its leading point. One frame of the resulting strobe is portrayed in Figure I.C.5. The display resembles symbols moving across an array of neon lights, and also the airport strobe lights which guide pilots down a runway. The strobe gets its name from this similarity.



(1)



(2)



(3)

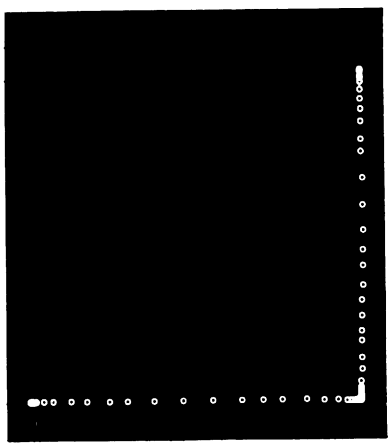
(1) The X coordinate waveform of a movement.

(2) The Y coordinate waveform of a movement.

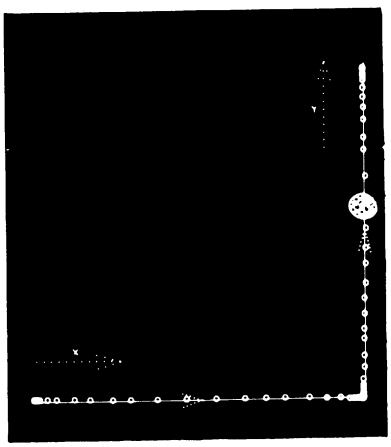
(3) A parametric curve representation of the same movement. The rhythm of the motion is not visible.

Figures I.C.1-3

PICTORIAL REPRESENTATIONS OF A CONTINUOUS MOVEMENT



(4)

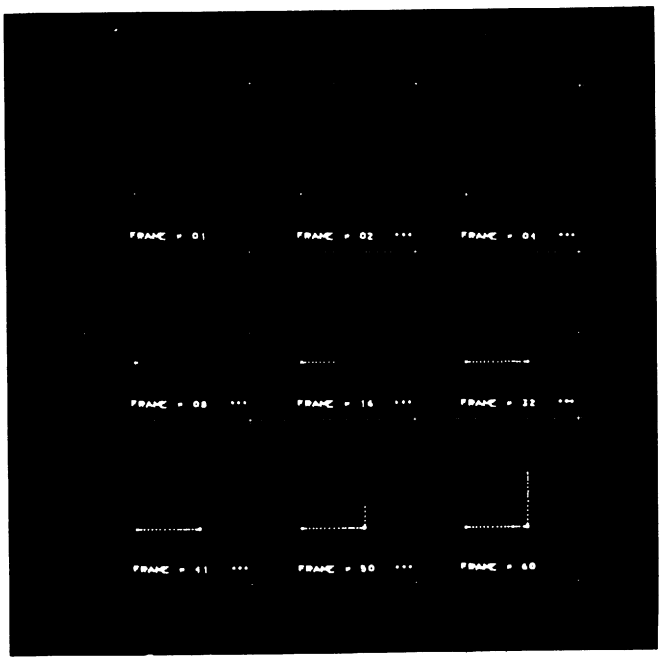


(5)

(4) A better display of the parametric curve of Figure I.C.3. Symbols are deposited at short, uniform intervals of time.

(5) One frame of the strobe corresponding to Figures I.C.1-4. The strobe is a dynamic display, in which superimposed on a static parametric curve is a marker tracing in real time the trajectory.

Figures I.C.4-5
THE PARAMETRIC CURVE AND ITS STROBE



(6) The p-curve corresponding to Figures I.C.1-5. The dynamic display is compressed into a single static picture containing nine selected frames, the 1st, 2nd, 4th, 8th, 16th, 32nd, 41st, 50th, and 60th.

Figure I.C.6
THE P-CURVE

I.C.3. PICTORIAL REPRESENTATIONS OF PATH DESCRIPTIONS -- AN ANALYSIS

The various pictorial representations of path descriptions should be compared and contrasted on at least the following criteria:

(1) their utility as mechanisms for defining new movements, and the quality of visual feedback provided the animator during this process;

(2) the number of dimensions of dynamic information that can be represented, and the ways in which the dimensionality is used to advantage;

(3) their role in guiding spatial and temporal adjustments to existing movements, and the naturalness and flexibility of the mechanisms that effect these changes; and,

(4) their capacity for meaningful conceptual extensions.

These criteria also apply to representations of selection and rhythm descriptions.

One major concern links our discussion of these issues. Recent experience with GENESYS has confirmed that a critical problem of picture-driven animation is that of coordinating various parallel actions, or concurrent dynamic activities. The relationships between actions, between movements, are more important than the character of each individually, for actions are always interpreted in context. With global

dynamic descriptions an animator can easily control entire intervals of individual movements. This chapter describes methods (better ones are still sought) for guaranteeing that desired constraints, or relationships between the movements of several objects, are satisfied, both at individual frames and over entire intervals of time. (*5)

Defining and Displaying Movements

Using static representation, such as a waveform, to define a new movement requires a thorough rationalization of the desired dynamic behavior, in other words either previsualizing or making analytic the changes of picture parameters with time. A waveform can also be used to depict the relationship with time of any picture parameter in an existent dynamic sequence. We have seen that these parameters may be spatial coordinates, scale factors, intensity variables, or line thicknesses. The display of a waveform highlights such regular behavior as a constant rate of change or such symmetries as periodicity or a perfect reversal of a given motion.

The p-curve is the direct mimicking of a movement's trajectory in an appropriately chosen space. The single action of the two-dimensional p-curve is equivalent to sketching two waveforms. Use of the p-curve facilitates a direct transmission to the computer of an intuitive feeling for a motion, of a bodily-sensed rather than visually-

rationalized dynamic. There are certain movements, such as a flutter, a tremor, and a thrust, which cannot be analyzed or formalized as easily as they can be mimicked with a hand motion. Emotions, too, are spontaneously conveyed by human motor behavior, as described by Arnheim in the following passage:

> Physiognomic movement is the component of bodily activity that spontaneously reflects the nature of the given personality as well as that of the particular experience at the given moment. The habitual firmness or weakness, confidence or timidity of a person is expressed in his movements. At the same time his bodily behavior will reveal whether he is interested or bored, happy or sad at this particular minute.
>
> Descriptive movements are deliberate gestures meant to represent perceptual qualities. We may use our hands and arms, often supported by the entire body, to show how large or small, fast or slow, round or angular, far or close something is or was or could be. Such gestures may refer to concrete objects or events --such as mice or mountains or the encounter between two people--but also figuratively to the bigness of a task, the remoteness of a possibility, or a clash of opinions. . . . (*6)

Display of a two-dimensional p-curve sometimes obscures precise temporal relationships, for it may be difficult to see how many frames separate pairs of points along a trajectory. The p-curve does depict clearly the temporal concurrence of the values of the two changing parameters which are represented. This is most useful for pairs of coordinates which are closely coupled intuitively, such as pairs of spatial coordinates of one object. The clearest static representation of the dynamics of a bouncing ball is a multiple-exposure,

high-speed photograph of the trajectory of the ball, and that is essentially the parametric curve.

Experience has shown that it is difficult and unnatural to conceive of two-dimensional spatial movements in terms of the two waveforms that are its formal equivalent. The point-by-point p-curve is a technique for defining such a motion directly in the plane, without being required to mimic it. Display of the strobe of an existing movement, as we shall see in I.C.7., should aid the mimicking of a new movement in relation to the existing one.

Occasionally one may wish to mimic dynamic behavior that consists of a variation with time of one parameter only. If the p-curve technique is used, only one of the resulting path descriptions is needed. In I.D. we depict the fluttering of a heart by rapidly varying its size. In a system with suitable algorithmic control, the animator assigns the X [R] coordinate of the p-curve to a parameter determining the size of the heart, and then flutters the stylus back and forth horizontally [radially]. Any vertical [angular] motion of the stylus is uninteresting and is ignored.

Dimensionality

A p-curve represents one more dimension than does a waveform. The stick figure experiment, discussed in I.E.2., established early in the research the utility of defining a two-dimensional movement with a single action. On the

other hand, two dimensions can only be defined jointly if they are closely coupled intuitively. Thus, we do not know if the added dimension is useful in situations other than that of pairs of spatial coordinates of the same object.

The p-curve allows a natural extension to three dimensions; one can trace a spatial path with a wand[56] as easily as a planar path with a tablet stylus. Sketching a spatial waveform of two coordinates and time, on the other hand, seems awkward, due to the difficulties in controlling the wand as well as the lack of a suitable related feedback mechanism.

The use of additional hardware graphically illustrates the power of the dynamic input of a multi-dimensional parametric curve. The Computer Image Corporation has built an electro-mechanical harness which records data about the bending of the joints (and hence the movement) of the man who is wearing it.[29] This allows the direct transmission of life-like dynamics, ideal for driving motions of a computer model of a human figure or an abstract stick figure. If music is synchronized by using it as the driving function for other dimensions of the movement, then the animation becomes even more energetic and life-like.

Refining existing movements

The waveform and the parametric curve each play a role in the spatio-temporal adjustments of existing movements.

Consider a display of the waveforms of a number of related parameters superimposed against a common time axis. Such a representation is natural for sequencing or pacing a number of concurrent movements. It has been useful to mark interesting events, or critical instants along the time axis, with a rhythm description. In the case where the coordinates of the p-curve do in fact denote spatial coordinates, spatial alignments of different objects are facilitated by use of their parametric curves.

Conceptual extensions

The concept of the waveform may easily be extended to other qualities of a motion such as the velocity. A similar extension of the p-curve does not appear to be useful. Although one could construct a p-curve plotting the changes of one velocity against another, it would be difficult to synthesize and conceive of a motion in these terms.

I.C.4. SKETCHING AND RESKETCHING PATH DESCRIPTIONS

Several observations may be based on accumulated experience sketching and resketching pictorial representations of path descriptions, both statically and dynamically.

Static Sketches (Not made in real time)

The fact that the functions representing waveforms must be single-valued may be exploited in the sketching process. One traces back and forth over a particular region of the curve, continuously replacing old values with new ones, thereby refining the shape of the waveform.

A particular problem arose in ADAM, the stick figure experiment, when sketching waveforms of rotation angles, whose maximum and minimum values (+180°,-180°) are considered equal. A curve can therefore go continuously from the greatest positive values to the greatest negative values. The animator should not have to break the continuity of his sketch in crossing an arbitrary boundary, such as 180°. A solution implemented in ADAM is that of identifying several regions of the display with a single range of values of the waveform. Specifically, while the animator continues sketching an effective 180°, 190°, 200° . . . a copy of this segment of the waveform appears at -180°, -170°, -160° . . . Only when it is convenient does he interrupt the sketch and continue it below. Commands for scaling and windowing pictures, augmented by good clipping mechanisms, would aid the application

of this technique. (*7) Such commands would also aid the resketching of path descriptions, for one could work on enlargements of particular portions of their waveform and parametric curve representations.

Another solution of more general interest, discussed but not yet implemented, is that of displaying waveforms so that the parameter being defined is represented by the identical parameter in the picture. In the case of rotation angles, the waveform is plotted in polar coordinates, R is identified with movie time in frames, and θ represents the angle to be varied. Values of the angle that will be used in successive frames of the movie are placed at increasing distance from the origin. Application of this technique, automatic in the construction of waveforms of Y versus time, might also be appropriate to the definition of X coordinate and scale factor variations. In the former case, the waveform is plotted from bottom to top instead of left to right, with the X coordinate of the display representing X, and the Y coordinate of the display representing movie time. In the latter case, R is identified with the size parameter, and θ with the time in frames.

We noted in I.C.3. that a parametric curve is an intuitively natural coupling of pairs of spatial coordinates, and hence more effective than a pair of waveforms as a vehicle for defining two-dimension spatial movements. New parametric curves may be constructed in real time, with a p-curve, or

not in real time, by positioning individual points of the curve. Currently in GENESYS, if only one dimension of a given movement need be altered, the corresponding waveform must be resketched. A new technique, which allows this to be done directly on the parametric curve, has been devised though not yet tested experimentally. Specifically, the method allows the animator to change the variation of x(t)[y(t)], while holding y(t)[x(t)] constant. Because a parametric curve can be multi-valued, the animator must designate the first and last points of the region of the curve to be resketched. Suppose, for example, that x(t) is to remain fixed, the varying stylus location is represented by (x_{s_i},y_{s_i}), the initial point to be altered is (x_1,y_1), and the next point along the curve within the designated region is (x_2,y_2). The stylus is set down on the surface of the tablet at (x_{s_1},y_{s_1}), and the first point of the curve is repositioned at (x_1,y_{s_1}). Sketching begins, and when the stylus coordinate $x_{s_2}=x_2$, the second point of the curve is moved to (x_2,y_{s_2}). Intuitively, points on the old curve are projected directly upwards or downwards to become points on a new curve. Since these coordinate pairs are the values of the X and Y path descriptions at successive frames, the variation of X with time is not altered. Because the animator must move strictly forwards and backwards between the boundaries of the delimited region of the curve, multi-valuedness is no longer a problem. The technique is formally equivalent

to sketching y(t) directly as a waveform. Intuitively, however, it is very different, because it uses a display which depicts correctly and naturally the coupling of X and Y.

Dynamic Sketches (Made in real time)

A problem in the sketching of p-curves is that of the synchronization of the hand to the computer in the beginning transient of the movement. One solution is the use of a countdown procedure. Another is to begin clocking before the start of the movement, and in subsequent editing to chop off the excess on the curve. In any case, editing can later be used to adjust the initial values assumed by the path descriptions.

We have noted that discontinuities in a p-curve can be introduced by lifting the stylus from the surface, since this temporarily halts the clocking of the movement.

It is difficult, when mimicking a motion, to control accurately its exact duration in frames. A satisfactory solution would allow the animator to sketch a p-curve without concern for its duration. If its length then fails to satisfy some other constraint imposed by the film, it can be scaled linearly in time to the desired length.

More generally, it is difficult to control accurately both the position and the speed of a movement. There appears to be, as one would expect, intrinsic constraints in the medium of hand motions. It is relatively easy, for example, to move quickly and accurately at regions of high radius of

curvature, and then slowly at regions of low radius of curvature. It is difficult, on the other hand, to do the converse. This observation, and other evidence to be introduced, suggests that the capability for dynamic mimicking must be augmented by an editing capability, a system for refining existing movements, if we are to achieve a flexible definition facility for a wide variety of dynamic behavior.

A technique has been devised, though not yet tested in practice, for combatting the difficulty in controlling both the position and the speed of a movement. The idea is to control each in separate motions. One would first sketch a p-curve without concern for the speed of the movement, concentrating only on the shape of the parametric curve. After the curve is suitably defined, we would retrace it with the appropriate dynamics, concentrating on the speed rather than on the exact position of the stylus. The system would then automatically project points from the new curve onto the old curve, presumably by a technique of orthogonal or minimum distance projection. Thus the precise spatial characteristics of the first movement would be combined with the dynamics of the second movement.

Another method, which we have used, is to concentrate first on mimicking the correct dynamics. We then adjust the spatial characteristics of the resulting curve with the editing system. This facility for refining existing movements is our next topic.

I.C.5. EDITING AND GRAPHICALLY REFINING PATH DESCRIPTIONS (*8)

Whereas the previous section presented methods for re-sketching a path description free-hand, both in real time and not in real time, this section introduces methods that do not require a sketch. Once a path description has been evaluated and judged not to be quite right, it is often easier to modify it rather than redraw it. These modifications, harking back to the distinctions of I.C.1., fall under category (2), refinements of existing pictorial representations, and category (4), indirect algorithmic specifications in terms of existing path descriptions.

We shall describe four major kinds of editing capabilities, operations for:

(1) scaling curves;

(2) reshaping them;

(3) algebraicly combining them; and,

(4) logically combining them, including pattern scanning, matching, and transforming functions.

Operations of the kind described in I.C.4. and I.C.5. should enable precise control to be exercised over the texture of a motion.

Scaling Curves

The need to scale path descriptions is accentuated by the difficulty noted above, that of controlling while sketching the precise magnitude and duration of spatial excursions.

Additive (multiplicative) scaling of the ordinate of a waveform is equivalent to the algebraic sum (product) of that waveform and a scalar or another waveform. Selective acceleration and decelerations of portions of a motion may be achieved by replacing "t" with a function "f(t)", which we call "generalized temporal scaling." A path description "x(t)" is thus mapped into "x(f(t))", the algebraic composition of the two time functions. All tools available for sketching waveforms should obviously be made applicable to the contruction and modification of scaling functions.

Our animation experience suggests that generalized temporal scaling is a technique more powerful than would in common practice be needed. A simple linear scaling command is exceedingly versatile if one can arbitrarily delimit the section of the path descriptions to be scaled. One would frequently choose a critical point on a waveform, and then shrink and expand by a given number of frames sections delimited on either side of the point.

Reshaping Curves

Scaling is the simplest example of the reshaping of path descriptions. More generally, one may take a curve, "attach" the stylus to one of its constituent points, and move it to a new position. A variety of algorithms can be used to compute how points adjacent along the curve are altered. For example, a vertical pull could define a deformation or

stretching whose effect diminishes (say, exponentially) on either side of the attached point. The decay constant, or stiffness factor of the curve, could be varied by a knob or by a pressure sensor in the tip or on the side of the pen. (*9) Alternatively, we may wish to maintain the trajectory of an existing two-dimensional movement, but vary the rate at which part of it is traced. If points along the parametric curve are viewed as beads on a string, we reposition some of the beads by designating one point of the string with the stylus, and giving it a quick vertical impulse. As before, we control the magnitude and extent of the reshaping operation by several numerical parameters, which in this model represent the mass of the beads, the strength of the impulse, and the effects of gravity and friction.

One would actually like to control reshaping so that certain constraints remain satisfied. For instance, if a particular point on a given curve defines a desired exact collision between two objects, then this point should remain fixed. Two other simple examples are requiring a curve to hold a given velocity in a certain region, and constraining it not to exceed a given ordinate value. The latter condition could be used, for example, to guarantee that one object would not penetrate another. One may wish to hold fixed the locations and heights of certain maxima and minima while rounding and smoothing, or sharpening and intensifying the peaks. One may also wish to make more or less continuous

and gradual certain level transitions.

There is only one implicit constraint mechanism currently in GENESYS, and it is heavily used. The animator may establish a Left Boundary and a Right Boundary, which delimit an interval of frames. Any activated sketching, resketching, or editing operation can only effect frames within the region currently delimited. This is a good way to localize effort within a particular interval of movie time. The feature also provides valuable protection against inadvertently altering an already perfected section of a movement.

Algebraicly Combining Curves

We have already seen the value of algebraic combinations such as the negation of a path description, which defines the reversal of a movement, and the composition of two path descriptions, which results in generalized temporal scaling. The sum of two path descriptions superimposes synchronized motions. For example, one generates a cycloid from the combination of a circular movement and a straight line movement. The difference of two path descriptions represents relative motion. In these examples path descriptions are interpreted as arrays of numbers, and algebraic operations are applied element-by-element to the values at corresponding instants of movie time.

Logically combining curves

Sections of path descriptions may also be viewed as

ordered strings of elements. Operations on strings such as cancatenations, insertions, deletions, periodic repetitions, delays (phase shifts), and head-to-tail inversions combine and extend existing dynamic sequences. Cancatenation causes one movement to follow after another. Insertion is used to insert one movement somewhere in the middle of an existing movement. An important special case in practice is the insertion of a string whose elements have a constant value equal to that of the element at the point of insertion, for this is a mechanism whereby holds, or pauses, are introduced into a motion. Deletion removes unwanted parts of an existing movement. The other operations may also be simply explained.

One could extend the interpretation of path descriptions as strings of symbols by providing a pattern-matching capability. Coupling this to standard scanning and pointing mechanisms would allow one to isolate aggregates of points satisfying criteria such as a constant velocity, accelerations greater than a prescribed limit, or a particular kind of discontinuity. The isolated aggregate could then be transformed in such ways as being smoothed or having a constant velocity transition introduced. Within this framework one could generate particular stylized kinds of dynamic behavior.

I.C.6. SPECIFYING SELECTION AND RHYTHM DESCRIPTIONS

Selection and rhythm descriptions are conceptually discrete phenomena; path descriptions, continuous. Whereas the elements of a path description are arbitrary real numbers, the elements of a selection description are small positive integers. A rhythm description may be viewed as a sequence of integer values of movie time, or as a Boolean sequence, a succession of ones that represent events and zeros that represent the absence of an event. Techniques for the construction of discrete dynamic descriptions are considerably simpler, yet conceptually similar to those used in the synthesis of continuous dynamic descriptions.

We have used various hardware devices in the static definition of selection and rhythm descriptions. Pictures of the former as a sequence of levels or steps, and of the latter as a sequence of pulses or event markers, may be drawn and accepted by suitable stylus input routines. Such pictures appear in Figures I.E.8, 17, and 18. It is often quicker to type rather than to draw a sequence of choice values, pulse positions, or intervals of movie time. Individual pulse positions may also be accurately positioned with a knob. Precision to the individual frame, often not essential with descriptions of continuous change, is usually critical here.

Like path descriptions, selection and rhythm descriptions

may be dynamically defined. What is required in the former case is the isolation of discrete positions or regions to which the stylus position in each frame can be assigned. One could for instance mimic in real time a line of music, moving the pen up and down as a baton on a hypothetical staff in the tablet surface. An appropriate algorithm would map this action into a selection description. A direct and simple dynamic input of a rhythm description consists of the tapping of a push-button. What more natural way to describe a beat such as "/ // / / /" is there than to actually mimic it to the computer ... "/ // / / /".

Tools for the editing of selection and rhythm descriptions are also required. The principles behind temporal scaling as applied to path descriptions are equally applicable here, although the analog to waveform shaping does not seem useful. All string operations carry over from the continuous descriptions to the discrete ones. There are element-by-element logical operations, such as "and" and "not", which correspond to the algebraic functions on waveforms.

A particularly useful discrete function is achieved by a simple coupling of a selection description and a rhythm description. A selection description is considered to be time-independent, if successive elements of the data sequence indicate only the order in which elements are to be chosen but not the time interval over which each choice is to apply.

Thus, if there are three states of the mouth, a grin, a smugness, and a frown, then the time-independent selection description S1 = (1,2,3,2,3) indicates only that the mouth undergoes the transition sequence grin-smugness-frown-smugness-frown. This description can be altered into a time-based selection description through the definition of a rhythm description R = (3,2,4,1,4), whose successive elements represent the intervals during which each state exists. A system which had a suitable command and implementing algorithm could transform S1 and R into the time-based selection description representing the frame-by-frame evolution of the mouth, S2 = (1,1,1,2,2,3,3,3,3,2,3,3,3,3). GENESYS contains, in crude fashion, a mechanism for doing this.

The concept of the dynamic pattern is suggestive when applied to discrete dynamic descriptions. Graphic representations of particular recurring movements, such as walking, kicking, and biting, appear as easily distinguishable patterns in the selection descriptions of Figure I.E.17. Periodic and almost periodic sequences of choices from a cel class are useful in animation. So are regular sequences of intervals between transitions or other events.

We often want to constrain sets of selection sequences. For instance, whenever the eye of a particular cartoon figure is closed, the lower eyebrow position would be the correct choice. Discrete analogs to smoothing operations are also meaningful. An example is the introduction of one frame of

position 2 (cel number 2) at every transition from position 1 to position 3.

smugness, and a frown, then the time-indep[illegible] description S1 = (1,2,3,2,3) indicates only that the mouth undergoes the transition sequence grin-smugness-frown-smugness-frown. This description can be altered into a time-based selection description through the definition of a rhythm description R = (3,2,4,1,4), whose successive elements represent the intervals during which each state exists. A system which had a suitable command and implementing algorithm could transform S1 and R into the time-based selection description representing the frame-by-frame evolution of the mouth, S2 = (1,1,1,2,2,3,3,3,2,3,3,3,3). GENESYS contains, in crude fashion, a mechanism for doing this.

The concept of the dynamic pattern is suggestive when applied to discrete dynamic descriptions. Graphic representations of particular recurring movements, such as walking, kicking, and biting, appear as easily distinguishable patterns in the selection descriptions of Figure 1.E.17. Periodic and almost periodic sequences of choices from a cel class are useful in animation. So are regular sequences of intervals between transitions or other events.

We often want to constrain sets of selection sequences. For instance, whenever the eye of a particular cartoon figure is closed, the lower eyebrow position would be the correct choice. Discrete analogs to smoothing operations are also meaningful. An example is the introduction of one frame of

I.C.7. COORDINATING PARALLEL ACTIONS

Two methods already presented are particularly useful for coordinating and synchronizing concurrent dynamic activities:

(1) Graphing dynamic descriptions along a common axis of movie time, which is depicted in Figures I.B.1, I.E.8, and I.E.17, aids the synchronization of one dynamic strand to another. (*10) In practice, we place event markers along the axis to aid the process of relating the values of several descriptions at one instant or interval of movie time. We can identify from the appearance of the selection description of the cel class 'mouth', for example, those intervals in which a character's mouth is in the fully open position. We then use this information to refine the selection description of the cel class 'leg', thus guaranteeing that his foot will be in the mouth soon after it is opened.

(2) Interacting spatial movements of several objects may be improved by graphically editing their parametric curve representations. Symbols on adjacent curves, when viewed in the correct correspondence, directly depict the relative spatial position of the objects in particular frames.

We can distinguish at least three other techniques which help achieve correct coordination:

(3) The attachment point of a cel is that constituent point whose coordinates are precisely defined by the path

descriptions which drive the cel. When sketching p-curves and parametric curves, the animator can better relate one movement to another if he can control and vary a cel's attachment point. For some movements of the wedge in 'SPROINGBOINGZAP' driving the center of the wedge is appropriate; for other movements, such as its striking the block, one should mimic the motion of the wedge's lowest point, since it and not the center makes contact with the block.

(4) We have developed an untested method for using the strobe to guide the dynamic mimicking of one motion in relation to another already defined. The technique is to play back the strobe of the existing movement while sketching the new p-curve. The static component of the strobe, the underlying parametric curve, should allow one to anticipate its trajectory and thereby better relate the new trajectory to it. The dynamic component of the strobe, the moving marker, should aid the synchronization of the rhythm of the new movement to that of the old movement.

(5) Often it is futile to work globally, over an interval of time, until a single critical frame is corrected. Such frames are usually transition points--instants of collision, beginning or final frames of movements, or pauses in an activity. Our limited experience suggests strongly that the ability flexibly to construct and adjust such individual frames is critical to the success of a picture-driven animation system. This is because it may be too difficult, and

it certainly has thus far seemed unnatural, to establish by global operations all the subtle interrelationships that occur in individual frames. After perfecting a key frame, however, we can better sketch the path descriptions that emanate forwards and backwards from it in time. Furthermore, we can sometimes interpolate path descriptions between two perfected frames.

This observation does not detract from the value of global dynamic descriptions and operations over them. It is simply a restatement of what should be obvious: they do not solve all the problems of animation. Global descriptions are useful when there is significant regularity in the changes of some aspect of a picture over an interval of frames. If there is no regularity from frame to frame, or if the relationships within a frame are more significant than the relationships from frame to frame, then it may be better to work individually on some or even all of these frames. It is too early to predict what mix of global and local operations would be used by an animator fluent with GENESYS and with the way of thinking developed in this dissertation. Ideally, he should work both globally and locally, each where most appropriate. Further details on local operations in GENESYS are given in I.E.4.

I.C.8. IMPLICATIONS OF THE FACT THAT DEFINING AND REFINING DYNAMIC DESCRIPTIONS IS AN ART

We recognize that this chapter has been rather vague about the relative importance of the many listed techniques, and the extent to which each is successful in aiding an animator working on actual animation sequences. Clearly, the experimental aspects of the ideas discussed above have thus far been explored to a limited extent only.

On the other hand, a few additional months of experimental work would still not yield precise rules for the usage of each specific technique. What constitutes the most effective set of tools for manipulating global dynamic descriptions depends heavily upon the artist, his range of experience with the medium, and the effects he is seeking to achieve. It is therefore essential that a picture-driven animation system be easily adaptable to the needs of the user. Let us analyze the meaning of the desired flexibility in a pair of important examples.

(1) A critical problem of picture-driven animation is relating the structure of cels (form, composition, balance) to the structure of the imposed dynamic behavior (movement and rhythm). This problem, which arises because picture-driven animation separates the definition of static and dynamic structure, is related to the difficulties of coordinating parallel actions. (*11) There are two aspects to the

problem: What information about cels and their interrelationship does an animator need to define successful picture dynamics? How should the animation system aid in making this information accessible and tangible, through providing appropriate displays?

More specifically, we ask: What should be visible when an animator is sketching or refining a path description? How can we combat the difficulty in utilizing the visual feedback provided by a p-curve's display while dynamically sketching that curve? Should the cel class under the control of the path description be concurrently displayed, or all "closely related" cel classes, or complete frames? Should this display be static, that is consisting of contents of individual selected frames, or dynamic, that is changing to show the frame affected by the animator's action at a given instant? Should representations of other dynamic descriptions be visible for comparison?

Thus, when the animator dynamically mimics a movement over the interval, frame a through frame b, what else should be visible on the display? A possible solution is: The current contents of frame (a-1) and frame (b+1) should be visible, for this will aid relating the p-curve to the structure of the frames just before and just after the new movement. However, display of these frames may be unnecessary if the object's motion is independent of other objects in the picture, may be unwise if the frames are so complex

that the scope becomes hopelessly cluttered, and may be inadequate if it is really the intermediate rather than the beginning or final states of the movement that must be closely coupled to other aspects of the scene.

(2) The second example deals with the naming of structures. We have already presented the need for flexible aggregation mechanisms for cels and cel classes. There is a parallel need for naming and manipulating arbitrary aggregates of global dynamic descriptions. Such aggregates would be useful when trying to relate movements of various cel classes.

How do we achieve the desired flexibility? In GENESYS, we continue to add new mechanisms and refine existing ones. To deal with the first problem area named above, we would provide GENESYS with commands for positioning and scaling individual pictorial representations of cels and dynamic descriptions, and commands for making these visible and invisible. To deal with the second problem area, we would provide GENESYS with commands for grouping and naming groups of dynamic descriptions, and extensions of existing commands so that they apply to such groups as well as to individual descriptions.

Thus we continue to add to the system new kinds of entities (new data types), and new commands which operate upon these data types. I contend that this approach is awkward

and inelegant because of the fragmentation of the resulting conceptual structure. GENESYS was not built with such extensions in mind, and so its growth is labored.

On the other hand, controlled and graceful extensibility may be achieved if we prepare for it by basing the system design at a fundamental level of abstraction, with the proper primitive structures and operators. Cels, dynamic descriptions, and animation sequences are not good primitives because they are too specialized. There are needed structures and operators, in terms of which the existing mechanisms of a system like GENESYS can be simply expressed, for then they can be made accessible for rapid modification and augmentation. Part II presents APPL, a new animation language designed with these issues in mind. The language is low-level enough to express the algorithms needed for picture-driven animation, yet high-level enough that the programs implementing these algorithms are relatively simple, and may therefore be changed with ease. In an animation system based on APPL, the animator could more easily comprehend how his system works and what is required to add new features and tailor it to his demands. The result would be a flexible, adaptable, multi-purpose animation machine.

FOOTNOTES -- I.C.

(*1) Notice that in approach (4) the data sequence is computed from existing data sequences, but in approach (5) it is computed directly from the movements of existing pictures.

(*2) It is hoped that APPL will be such a system, once it is implemented, provided with a variety of these techniques through definitional extension, and augmented by capabilities to accept direct analog input.

(*3) If the X coordinate of the stylus movement is identified with movie time, then the movement cannot also be sketched in real time, for the value of movie time obtained from the stylus position will rarely equal that obtained from the value of real time. We could, however, augment a p-curve with two growing waveforms appearing on another portion of the scope. In this case, the user would still be generating a p-curve, and not sketching a waveform.

(*4) I am indebted to Dr. W. R. Sutherland for suggesting the concept of the strobe.

(*5) Using frame-by-frame techniques, the animator works on each individual frame and thereby satisfies constraints local to a frame. In picture-driven animation, he works directly on sequences of frames, exercising global control over entire intervals of time. This economy of effort must not be achieved at the expense of loss of control over the relationships between objects in a scene.

(*6) Arnheim,[60] pp. 165-166.

(*7) 'Scaling' means enlarging or reducing in size. 'Windowing' means isolating and viewing a scaled version of a region of a picture, and usually implies that one will scan parts of the picture by moving the 'window'. 'Clipping' means removing from the display those portions of lines and curves that fall outside the window.

(*8) The term 'to edit' is used in the broad sense of 'to correct, to prepare, to arrange', and not in the narrow sense of 'to rearrange sections of photographic film', as is often done by splicing them.

(*9) Work on the shaping of surfaces in 3-space by Steve Coons,[61] and a related exploratory study by Tim Johnson,[62]

support the claim that such tools are useful. None of these have yet been implemented in GENESYS.

(*10) There is a strong analogy between displaying a group of dynamic descriptions on a common time axis and what is called a Work Book in commercial animation. The Work Book is a linear sequence of blocks, each containing information about a successive unit (such as half a second) of the movie. It is described in Halas and Manvell,[2] pp. 171-2:

> The work Book is derived directly from the final storyboard [the visual presentation of the idea of the film in a series of sketches], and is an analysis of each shot and sequence on a frame-by-frame basis. In this way the exact timing of every individual movement is determined, and its inter-relation with all other movements fixed at any given moment.
>
> The Work Book must also show what in a live-action film would be called the camera movement--any movement, that is, which is the equivalent of a tracking-in, panning or tilting shot.
>
> It must also show how each shot or sequence is to be punctuated, whether by a straight cut, a fade or a dissolve.
>
> The creation of the Work Book is the responsibility of the director. . . .

(*11) See I.C.3., particularly Footnote 5, and I.C.7.

I.D. A DETAILED EXAMPLE OF PICTURE-DRIVEN ANIMATION-- MANY SYNTHESES OF A PULSATING HEART

The purpose of this chapter is the clarification, through concrete illustration, of the approaches to the specification of picture dynamics that were introduced in I.B. We elaborate numerous techniques, mostly falling in the domain of picture-driven animation, for animating a pulsating heart.

The various techniques may not produce precisely identical sequences of frames, but they produce essentially the same movie, that is movies which convey the same general feeling. We stress how each technique facilitates a certain kind of control over the texture of the resulting dynamics. The example has been chosen for the purpose of explanation and not for its visual interest; such a variety of animation techniques are not required for a sequence this simple. Nonetheless, the techniques presented are useful in many other situations, and, despite their number, do not exhaust the possible methods of construction.

Each technique is summarized by listing the information, that is algorithms, pictures, and global dynamic descriptions, which must be supplied to define the sequence. What is done directly by the animator is distinguished from what is embedded in the system. Most dynamic data sequences are represented by static (single) pictures or dynamic (sequences of)

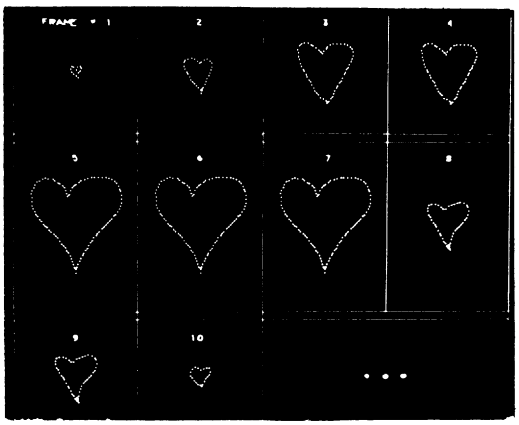
pictures. Conventions used in the diagrammatic representation of each construction method are explained as they are introduced.

Approach 1 -- The Individual Construction of Frames:

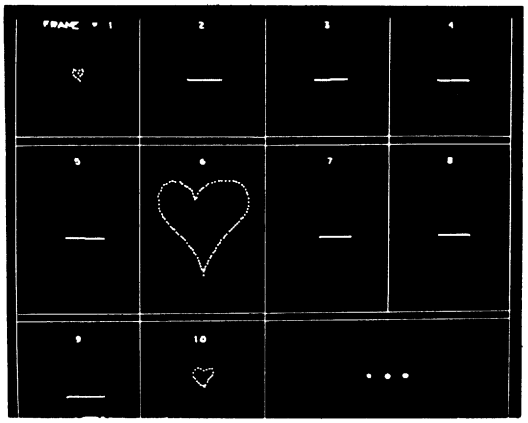
Construction proceeds by directly sketching each new element of the sequence. The result is precisely what appears in Figure I.D.1. The "..." indicates that the tenth frame is followed by arbitrarily many others. To save work without altering the effect of the sequence, we repeat some frames; hence, there are only six new pictures in that part of the sequence shown in the figure. There is no fixed or predetermined relationship among the sizes and shapes of the first, second, third, fifth, eighth, and tenth frames. The animator has total flexibility in exercising dynamic control, for each frame may be varied independently, but he also must assume the total burden of picture construction.

Approach 2 -- The Interpolation of Sequences of Intermediate Frames:

Direct sketching is used to create certain critical frames of the sequence, as is shown in Figure I.D.2. The blank mark [-] indicates the temporary omission of a frame. An interpolation algorithm, provided by the system, fills in the blanks. The flexibility of dynamic control depends upon the number of frames sketched by the animator and the variety of interpolation algorithms that exist in the system. (*1)



(1)



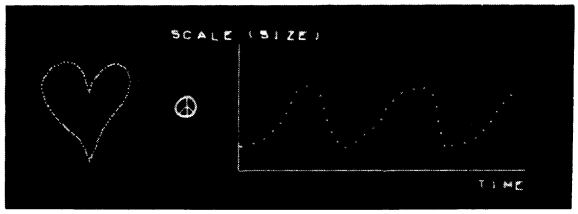
(2)

(1) The Individual Construction of Frames.
(2) The Interpolation of Sequences of Intermediate Frames.

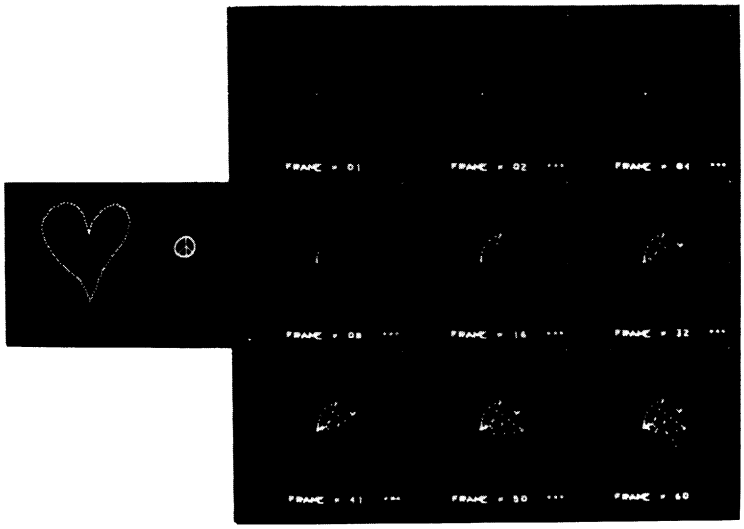
Figures I.D.1-2
MANY SYNTHESES OF A PULSATING HEART

Approach 3 -- The Generation of Frames from an Algorithmic Description:

The animator writes, in some language, a program which calculates the individual frames of the sequence. (*2) One approach he might take is to factor the construction into a form-generating algorithm and a movement-generating algorithm. (*3) One program could compute the heart-shaped mathematical function called a cardioid. Another program could compute a path description, perhaps a sawtooth function, to vary the size of the cardioid. All frames would then be scaled versions of a single heart shape. Within this approach, the animator has any kind of dynamic control that he can formally and precisely, that is, algorithmically, specify.



(3)



(4)

Picture-Driven Animation--Construction of a Cel and:

(3) a Waveform.

(4) a P-curve.

Figures I.D.3-4

MANY SYNTHESES OF A PULSATING HEART

Approach 4 -- Picture-Driven Animation:

Technique 1 -- Construction of a cel and a waveform, which is a static representation of a path description:

The animator sketches a single heart shape, and a waveform which defines the scale of the drawing. These are shown in Figure I.D.3. The symbol "☮" located between them signifies that the system combines the heart form and the waveform to generate an animation sequence. This is done by an algorithm, embedded in the system, which is capable of varying continuously the size of the cel. (*4) All frames are therefore scaled copies of the single heart. Dynamic control is exercised by refining the shape of the static structure, the cel, and the shape of the dynamic structure, the waveform. Techniques described in I.C. are used for the construction and modification of the waveform.

Technique 2 -- Construction of a cel and a p-curve, which is a real time representation of a path description:

Technique 2 differs from Technique 1 only in that the variations of the size of the cel are defined by another method. Specifically, the scale factor is sketched in real time as the radial coordinate (R) of a parametric curve. The p-curve, which is the dynamic construction of the parametric curve, is shown in Figure I.D.4. Successive values of the scale factor are determined by the changing distance of successive new points of the p-curve from the origin of the coordinate system. (*5) Notice how the curve, of which frame numbers 1,

2, 4, 8, 16, 32, 41, 50, and 60 are shown, swings out and in, out and in. The heart will consequently pulsate large and small, large and small. Dynamic control is exercised by mimicking one's intuitive feeling for the movement, and by employing other techniques discussed in I.C.

Technique 3 -- Construction of a cel class and a time-based selection description:

The animator sketches three hearts of differing size and shape. He specifies which of the cels is to appear in each frame by constructing the graph of a selection description. The graph, which appears with the cel class in Figure I.D.5, plots the choice of one of the three hearts against movie time. The system combines the display of static picture structure on the left with the display of dynamic structure on the right to form a movie. The image on the right drives the images on the left into motion. Dynamic control is exercised by altering the cels or the rhythm of switching from one cel to the next.

Technique 4 -- Construction of a cel class, a time-independent selection description, and a static representation of a rhythm description:

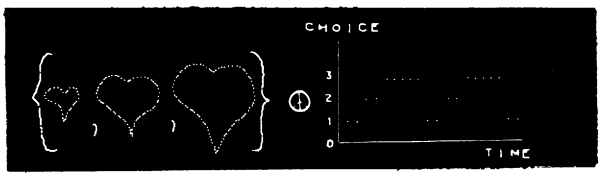
All frames are again selected from the same set of three hearts. This time we decompose the dynamic behavior, represented by the selection description, into the "product" of a time-independent selection description and a rhythm description. The former is an ordered list of choices from the

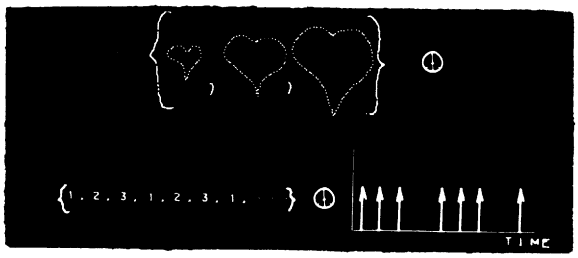
heart set; the latter determines the intervals of time over which each successive choice is active. The rhythm description is depicted by a static display of a pulse train. Each pulse triggers the replacement of the current image with the next choice on the list. Thus the animator specifies and the system combines the three components of Figure I.D.6. The representation shown in the figure defines an animation sequence precisely identical to that produced by Technique 3. The decomposition facilitates the exercise of independent control over the order of appearance of cels and the rhythm of their appearance.

Technique 5 -- Construction of a cel class, a time-independent selection description, and a real time representation of a rhythm description:

This technique is identical to Technique 4, except we do not use the static pictorial representation shown there to define and refine the rhythm description. It is instead dynamically defined, tapped out in real time on a push-button. The reader can simulate this representation by covering the pulse train of Figure I.D.6. with a piece of paper, and then pushing the paper from left to right in real time. As various points along the horizontal axis are reached, such as the first, third, and fifth frames, new pulses appear in the picture.

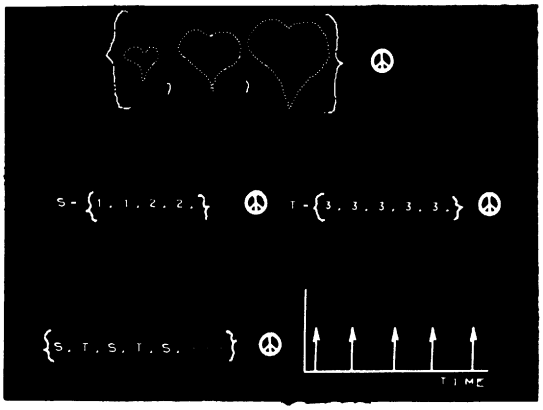
(5)





(6)

(7)



Picture-Driven Animation--Construction of a Cel Class and:
(5) a Time-Based Selection Description.
(6) a Time-Independent Selection Description and a Rhythm Description.
(7) a Dynamic Hierarchy.

Figures I.D.5-7
MANY SYNTHESES OF A PULSATING HEART

Technique 6 -- Construction of a cel class and a dynamic hierarchy:

All frames are again selected from this set of three hearts. The movement is determined by a dynamic hierarchy, a combination of three selection descriptions and a rhythm description. (*6) The result is precisely equivalent to that achieved by Techniques 3, 4, and 5. The animator specifies what is shown in Figure I.D.7:

{S,T,S,T,S,...} is a time-independent selection description which specifies that the animation sequence consists of an alternating selection from two subsequences of images, labeled S and T.

The rhythm of the alternation is defined by the rhythm description, where each pulse triggers the change from activity S to activity T, or from T to S.

{1,1,1,2,} and {3,3,3,3,3,} are time-dependent selection descriptions which define the two activities, S and T. S represents the activity of the expansion of the heart, the progression from the small to the medium sized heart, while T represents its holding the fully expanded position, the continued display of the largest sized heart.

The decomposition into this dynamic hierarchy enables independent dynamic control over several levels of the nature of the picture change as well as over its rhythm. It would be easy, for example, to modify activity S and therefore what it means visually for the heart to expand, without altering

any other aspect of the sequence.

The rhythm description appears to be redundant. This is not so, given the following generalization: If a pulse arrives before all images of S or T are displayed, the activity is terminated. If a pulse has not arrived by the time all images have been displayed, the last image remains visible until it arrives.

FOOTNOTES -- I.D.

(*1) GENESYS contains no interpolation algorithms. APPL would gracefully accept the embedding of such algorithms.

(*2) This can be done in APPL, but not in GENESYS.

(*3) Note the parallel between this decomposition and picture-driven animation's fundamental separation of static picture structure from dynamic descriptions.

(*4) Since GENESYS currently lacks this algorithm, we have simulated its effect by hand.

(*5) See Footnote 4.

(*6) Dynamic hierarchies cannot yet be established in GENESYS. Scale changes controlled by path descriptions (Techniques 1 and 2), p-curves whose coordinates may be identified with arbitrary picture parameters (Technique 2), and dynamic hierarchies (Technique 6) may be easily implemented in APPL.

FOOTNOTES -- I.E.

(*1) GENESYS contains no interpolation algorithms. APPL would gracefully accept the embedding of such algorithms.

(*2) [illegible] **EXPLORATORY STUDIES IN PICTURE-DRIVEN ANIMATION**

(*3) Note the parallel between this decomposition and picture-driven animation's fundamental separation of static picture structure from dynamic descriptions.

(*4) Since GENESYS currently lacks this algorithm, we have simulated its effect by hand.

(*5) See Footnote 4.

(*6) Dynamic hierarchies cannot yet be established in GENESYS. Scale changes controlled by path descriptions (Techniques 1 and 2), p-curves whose coordinates may be identified with arbitrary picture parameters (Technique 2), and dynamic hierarchies (Technique 6) may be easily implemented in APPL.

I.E.1. CONSTRUCTION, VIEWING, AND FILMING MOVIES ON THE TX-2

Three special-purpose picture-driven animation systems have been implemented on the M.I.T. Lincoln Laboratory TX-2 computer. A common feature is that each has a construction or editing mode, a playback or viewing mode, and a filming mode.

In the first mode the animator may begin work on new pictures and global dynamic descriptions, or may recall and continue the construction of pictures and descriptions saved from other sessions. Algorithms embedded in the systems then compute display files, in which sequences of frames composed of points, lines, and conic sections are encoded for display by a TX-2 cathode ray tube.

The image files are displayed as they are computed. They may then be repetitively displayed by the playback program, which simulates a variable-speed, bi-directional, video tape recorder. (*1) The program normally sequences through the display file representation of successive frames, making each in turn visible for 1/24th of a second. Continuous cycling through the movie, or the simulation of a tape loop, is one useful option that may be requested. The animator may at any time set or alter the direction and speed of playback by flicking toggle switches located next to his console.

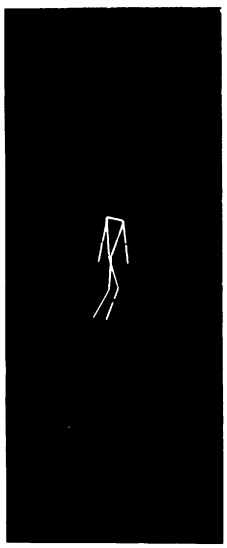
When the animator has prepared a satisfactory sequence, he need no longer view it directly on the scope, but may instead want to record it on film. A pin-registered movie camera can be mounted in a light-tight box to a TX-2 scope. Its shutter is always open. The filming program (a variant of the playback program) "paints" an image on the scope. After a sufficient time interval to allow the decay of the phosphor, approximately 1/5 of a second, a signal from the computer advances the camera. A return signal upon the completion of the advance triggers the display of the next frame. The camera can be operated on one scope while we work at a tablet with another scope. Excellent film quality, with high contrast and low jitter, can be produced with the system. We can expose film at 4 or 5 frames per second, a rate limited by the decay time of the phosphor. Since the TX-2 presently has no color scope, black-and-white movies are produced. If full color were required, this may be obtained by the generation of three copies of each frame, one containing the red information, a second containing the blue, and a third the green, followed by the expensive photographic process of optical superposition through color filters.

I.E.2. ADAM

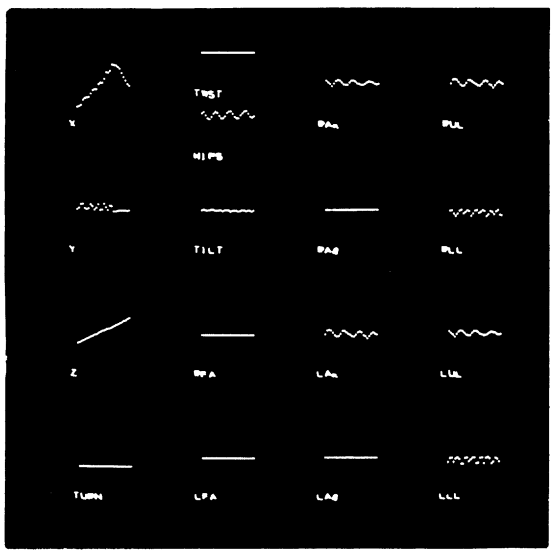
The chronologically first system, A Demonstration Animated Man (ADAM), allows one to animate a crude line-drawing representation of a single human figure. This is done by specifying, via waveforms and p-curves, the seventeen path descriptions that define the temporal behavior of the figure's seventeen controlling continuous parameters. These are the three coordinates of the position of the body center and various rotation angles that determine bending of parts of the torso and the limbs. A model of the stick man moves in a hypothetical three-dimensional space; what is displayed is a two-dimensional projective transformation of the figure.

The ADAM system in March of 1967 contained tools for specifying waveforms only. Initially all seventeen parameters were set to zero for all time, resulting in a lifeless man standing stiff and at attention. As each waveform was successively sketched and refined, the figure began to move. Complexity was added gradually--first translational motion, then leg motion (to achieve a crude walk), then hip, torso, and finally shoulder and arm movements. Alternatively, one could begin with a particular motion (set of waveforms), and alter it to form a related motion.

Surprisingly life-like walking, leaping, and flying motions have been synthesized. Figure I.E.2. shows the set of seventeen waveforms defining the movement of a figure who takes five leaps forwards and then suddenly finds himself on



(1)



(2)

(1) The stick figure. That he has no head is purposeful--the computer did not "drop a bit."

(2) Seventeen waveforms defining the movement of a figure who takes five leaps forward and then suddenly finds himself on a horizontal conveyor belt moving backwards. The curve in the top left represents his horizontal motio plotted against time, the curve below it, his vertical motion. The second curve from the top in the second column represents the swinging hip movement. Motion of the legs is defined by the four curves in the fourth column.

Figures I.E.1-2
ADAM

a horizontal conveyor belt moving backwards. One frame from this motion is shown in Figure I.E.1. The synthesis of such an action was possible despite the crudeness of the system and even though the figure was controlled by what has proved to be an awkward and excessively large set of parameters. (See below.) Consequently beginning efforts at constructing movements were incredibly painful. With only a tool for individually sketching and resketching each waveform (superimposed, if so desired, on plots of other waveforms), and with no prior thought or preparation, synthesizing and refining a motion such as the one in Figures I.E.1-2 could take several hours of console time. The task became somewhat easier with the addition of tools for copying waveforms, and for repeating them periodically and shifting them through time, thus aiding the introduction of symmetry and phase relations into the figure's movement.

Movement definition in ADAM was also facilitated by the addition in December of 1967 of a p-curve capability. The translational motion of a series of leaps could be obtained merely by mimicking the leaps with the stylus. This was in fact the technique used in the animation sequence of Figures I.E.1-2.

The most interesting observation was the result of an accident. Just for fun, I applied to ADAM several random collections of path descriptions (waveforms), defined by the p-curve technique for use in animating EVE. Only individual

fragments of the resulting motions were realistic; the sequences as a whole were energetic, wild, and nonsensical. Yet almost all movements appeared fully alive and life-like. Viewers perceived the movements as life-like even though they were obviously not physically realizable. This life-like quality was not apparent when all path descriptions were obtained by sketching waveforms.

I regard this result as rather striking visual evidence of the power of clocked hand-drawn dynamics, more fundamentally, of the approach of the dynamic mimicking of animated behavior. The result is also not as surprising as it may seem. If the same physical, physiological, and neurological principles determine the "style" or "character" of the motions of all human limbs, then the possible dynamics of a hand movement would be "similar to" the possible dynamics of a leg movement. A mathematical model might attempt to abstract the differences in hand and leg movements by different constant parameters of equations that otherwise have the same form for both limbs. A paradigm for this approach is the work of Mermelstein and Eden,[63,64] who, considering only the process of handwriting, discovered that:

> "It is found that the strokes of the writing may be classified into topologically similar categories generally independently of the writer producing the strokes. . .the representation of the letter in terms of stroke category sequences may be made independent of the writer as long as small variations are allowed for the average parameter values of each category." (*2)

The technique of editing and refining existing waveforms has proved to be a sensitive and powerful one. Slight modifications to waveforms resulted in significant alterations to the character of an extended interval of a movement --a normal walk was made into a jaunty saunter by the addition of more bounce to the vertical coordinate path description. An animator working with traditional media must laboriously "reconstruct" each frame if he is to make such a modification. The addition of one more command to ADAM would allow one to accelerate or decelerate any "dimension" of the movement. Hence I stress once more that the use of global dynamic descriptions in representing movies may reduce the amount of repetitive labor in animation, for it may be quicker to modify an existing sequence through its set of movement descriptions than to begin construction of a new one. Another example of such modification, taken from the use of GENESYS, is depicted in Figures I.A.5-10.

Work with ADAM also demonstrated that the following should be closely intertwined--the model of static picture structure, the choice of representation of picture dynamics, and the tools for the specification of dynamics. We noted before that the choice of parameters for the model of the stick figure has proved to be an awkward one. ADAM always had trouble keeping his feet on the ground--he had unknowingly been designed to fly. Specifically, there was no easy way to anchor his foot to the ground at the appropriate times during

a stride. The effect could only be achieved by laboriously adjusting the forward motion of the body, the swinging of the hips, and the rotational motion of the legs. This is because coordinates of parts of the body were measured relative to the body's center. If it were possible to define a coordinate of the foot directly, then it would be easy to anchor it. The particular choice of ADAM's parameters were well-suited for constructing swinging arm motions, but were ill-suited for describing a movement of stretching out the arm to retrieve an object, for the same reason that they were poor for anchoring the feet.

The "best" choice of parameters also depends upon the set of techniques available for the specification of their path descriptions. We recall from I.C. that p-curves can be usefully applied to pairs of rectangular coordinates from a particular joint or part of a figure. P-curves could have better been exploited by changing the static picture model so that the hip position relative to the foot were used instead of the two rotation angles that defined the two degrees of freedom of a leg motion.

More fundamentally, this reduces to a constraint satisfaction problem. A viable special-purpose solution would be to build a stick figure system in which leg motions could be defined in terms of variations of a redundant set of parameters, including rectangular coordinates of hip, knee, and foot, as well as rotation angles. The system would allow

the assignment of path descriptions to parameters over intervals of time, and would evaluate the constraint situation (overconstrained and inconsistent, consistent, or underconstrained and hence not well defined) existing in each interval. This computation would be straightforward for the stick figure model used in ADAM.

In conclusion, it is obvious that inputting stick figure motions by the sketching of 17 waveforms (or fewer p-curves) is a technique vastly inferior to mimicking a movement while wearing the Computer Image harness described in I.C.3. Yet the harness is only a transducer which maps a real human motion into a collection of waveforms, and must be supplemented with the capability to manipulate and modify the waveforms. The ability to transform existing dynamic behavior and distort it at will is essential, because <u>many cartoon motions cannot be mimicked or only so with difficulty--they are purposeful exaggerations and caricatures of real movements.</u> Halas and Manvell explain this in detail:[3]

> "The difference between cartoon animation and real life lies in the application of certain principles which are most appropriately called aesthetic.
>
> ". . .
>
> "As the figures in cartoon films begin to approach actuality more closely, the physical laws affecting their counterparts must be more strictly applied, even though they are still fantastically caricatured. The graphic beings in U.P.A.'s films . . . maintain an existence which is far from actuality in their quaint and angular outlines. Their movements are simplified, mechanized, grotesque. But they are funny because they nevertheless bear a direct relation to the human beings

and animals that they caricature. Their legs may rotate like cog-wheels, their bodies may glide and wave like cloth--but through these grotesqueries of movement shines the light of human observation, the comic realization of the problems of human mass and weight, gravity and friction, the physical laws of nature. The aesthetic and the physical meet on a sublimated plane of comic imagination. . . .

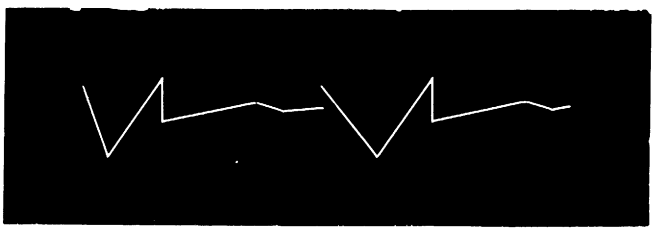
". . .

". . . It is fruitless, uneconomic, inartistic and very exasperating to undertake the laborious process of animation in order to tell a story in a manner which could be achieved just as well or even better through a live-action film. A cartoon's virtues lie in simplification, distortion and caricature. It distorts for its own purpose and affects both character and behaviour. It observes the physical laws of nature only to defy them. It conforms wholly to naturalism at its peril, because it then invites comparison with something fit to be reproduced only by the live-action camera. As the animator draws away from naturalism the powers of his medium increase; there is nothing but the limits of his imagination and his technical resources to hold him back." (*3)

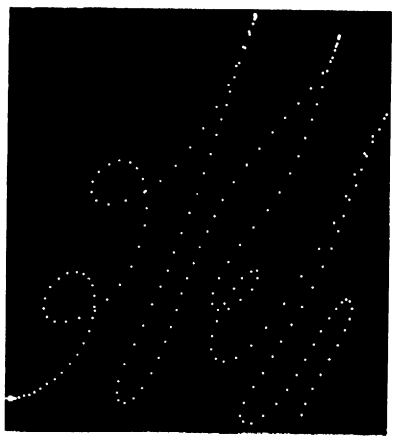
I.E.3. EVE

The chronologically second system, Evolving Visual Energy (EVE), allows one to animate in two dimensions a set of points linked by "rubber band" straight lines, which we call the "serpent" or snake." Except for the picture being animated, the EVE system is identical to the ADAM system, for it was in fact grafted and grown out of the latter. Seven points move together under the application of one planar motion, defined by one p-curve or two waveforms. Each point also executes an independent relative motion. These are defined by seven p-curves, or fourteen waveforms. Thus the movement of each point is the sum of the movements produced by the main driving function and its own individual driving function. The seventeenth waveform defines the scale, or size, of the snake and the scale of the local driving functions. It is also possible to couple the relative motions of the points.

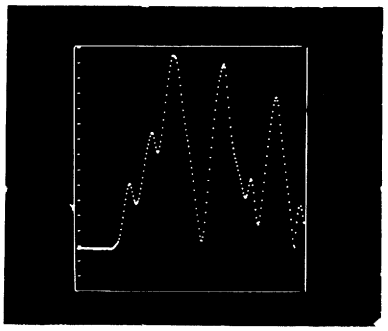
EVE is an exercise in abstract dynamic art constructed to illustrate the simplicity and power of applying a dynamically sketched parametric curve, or p-curve, to define a movement. The motion of the stylus is immediately mirrored in some aspect of the dynamic of the snake. One is in some sense executing a "sculpture in time," or generating dynamic sculpture. Complex, interesting, and "beautiful" movements have been achieved with a system again limited to the manipulation of path descriptions. A movement appears to exist in



(3)



(4)



(5)

(3) A pair of snakes (rumored to be preparing for a long sea voyage with, among others, a pair of crocodiles).

(4) A typical parametric curve (the last frame of a p-curve) involved in the definition of a snake motion. Notice how smooth and graceful the movement is. Observe how points cluster at pauses in the motion.

(5) A waveform depicting the temporal behavior of one coordinate from a p-curve similar to that shown in Figure I.E.4.

Figures I.E.3-5

EVE

three dimensions even though it has been generated in only a plane. Figure I.E.4 shows a typical p-curve and Figure I.E.5 shows a typical waveform involved in the definition of one of EVE's motions. One frame from the sequence, in which EVE appears twice, is shown in Figure I.E.3.

I.E.4. GENESYS -- THE GENERALIZED-CEL ANIMATION SYSTEM

The chronologically third system, unlike the first two, is more than an experiment. Although it too is a special-purpose animation system, it is versatile enough to be used in the generation of a broad class of two-dimensional dynamic images. This section fills in remaining details about the operation of GENESYS that have not been discussed in previous chapters.

GENESYS contains three basic types of commands, those that:

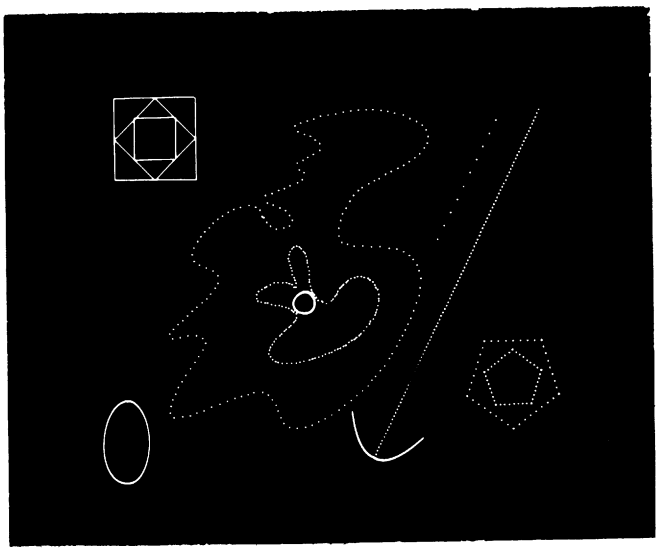
(1) Generate and modify static pictures;

(2) Define and refine global dynamic descriptions; and,

(3) Save and restore static and dynamic picture data, structure the display of existent pictures (including their playback), and set and change various modes of system operation.

Unlike ADAM and EVE, GENESYS allows the animator to construct his own static images by sketching them free-hand.

Two of these modes affect the meaning of all operations upon static pictures, frame-mode, in which each picture is assigned to a unique frame, and cel-mode, in which each picture is assigned to a unique cel in a unique cel class. Specific commands set the mode and the current pictorial context of the system. Thus, following the command "*EDITFRAME* 5;", any newly-sketched picture is made part of



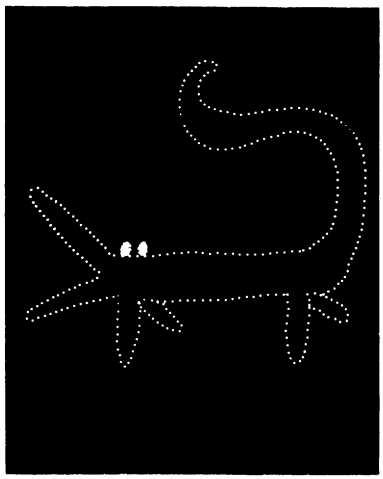
This picture, "drawn" by the author, illustrates the variety of line and texture that may be included in a GENESYS cel as of December, 1968. Free-hand sketches are portrayed by points spaced at an arbitrary, user-controlled density. Straight lines can be solid or can be dotted, over the same range of densities. Sections of circles, ellipses, parabolas, and regular ploygons may be included. Arbitrary subpictures may be copied, translated, rotated, and scaled along two independent dimensions.

Figure I.E.6

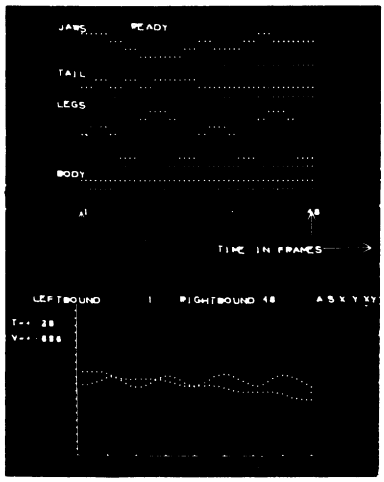
PICTORIAL FRAGMENTS FROM GENESYS

the fifth frame. Furthermore, following the command "*FORMCEL 5 in the class* P.HEAD;", a new sketch is assigned to the fifth cel in the cel class named 'P.HEAD'. A selection description associated with the cel class determines in which frames the cel is visible. All pictorial operations, such as additions, deletions, and transformations, are essentially identical in frame-mode and cel mode, but apply to the current pictorial context associated with the active mode.

The animator may sketch, copy, translate, rotate, reflect around axes, and scale two-dimensional subpictures consisting of points, straight lines, and conic sections. Freehand sketches are displayed as a trail of dots whose density is under user control through a number of toggle switches. He may erase by "picking up," or pointing to, individual picture elements with the stylus. Alternatively, he may request that the last pen stroke of a sketch be deleted or the entire context cleared. A variable-density rectangular grid is an aid to constructing regular line drawings. If the grid is activated, then the endpoints of all new straight lines will be constrained to lie on grid locations. The construction of the solid and the dotted straight lines in Figure I.A.13. was simplified by this feature. Arbitrary projective transformations may be directly applied to cels. All these capabilities are operational in the current version of GENESYS, and are illustrated by Figures I.A.13. and I.E.6. The techniques used are standard and well-known in the inter-



(7)



(8)

(7) The crocodiless hops across the screen, delighted with her recent creation at the TX-2 console. The artist, Miss Barbara Koppel of Chicago, had little animation experience, no computer experience, a brief introduction to GENESYS, and assistance in using it from the author.

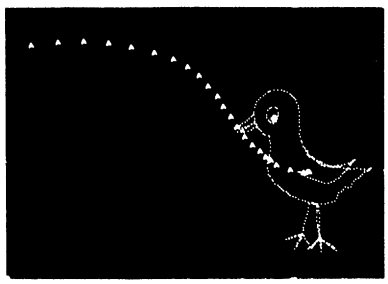
(8) The four selection descriptions generate the movements of the jaws, tail, legs, and body of the crocodiless. Her translational motion is defined by the two path descriptions below. The oscillatory waveform is the vertical coordinate, the waveform sloping downward the horizontal coordinate.

Figures I.E.7-8
SYNTHESIS OF A CAVORTING CROCODILESS

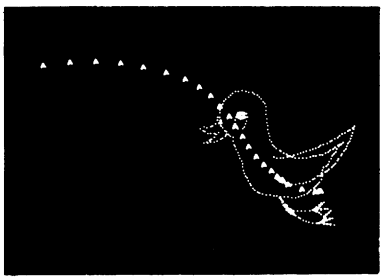
active computer graphics of still pictures.

GENESYS also contains a limited number of tools, patterned after the discussion of Chapter I.C., for the specification and manipulation of static and dynamic pictorial representations of global dynamic descriptions. P-curves may be generated in real time with the stylus; rhythms may be mimicked in real time with a push-button. The system currently expects construction of two path descriptions and one selection description to drive into motion each activated cel class. It is customary to establish a rhythm description marking interesting events by aligning vertical lines with the correct frames using a knob. GENESYS then replaces the lines with arrows such as appear in the middle of Figure I.E.17. The complementary role of cel classes and movement descriptions in the synthesis of two short animation sequences is illustrated by Figures I.E.7-12.

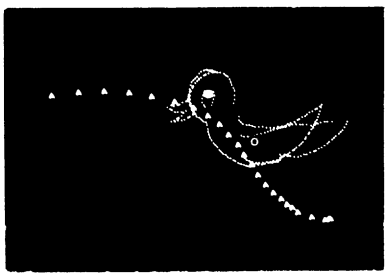
The refining of global dynamic descriptions often produces a movement with the desired quality globally, that is, over the duration of the sequence, but with occasional errors locally, that is, in individual frames. For example, the overall bounce of a ball may be correct, but its location at the instant it hits the ground may require modification. Local adjustments may be made in frame-mode, by calling up one particular frame and adjusting the values of the driving path descriptions in that instant only. The animator can also experiment with alternatives to the



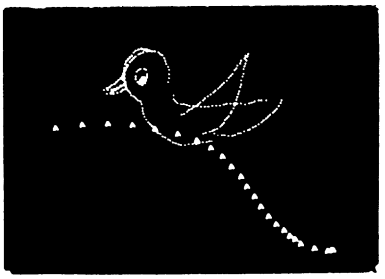
(9)



(10)



(11)



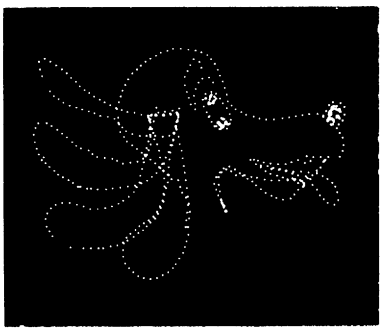
(12)

The 1st, 7th, 13th, and 19th frames of the take-off of a bird are shown. The figure is superimposed on the parametric curve (the final frame of the p-curve) which defines its path through space. The animator, Mrs. Nancy Johnson of Waltham, Mass., has mimicked the motion by sketching the p-curve; the bird then reproduces this movement. Observe the switching among discrete shapes and positions of its eye, wing, and feet.

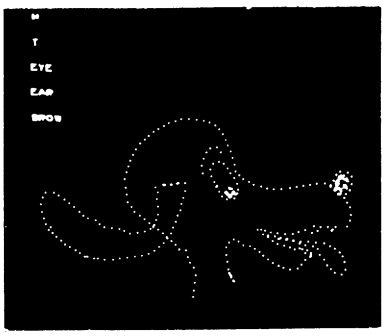
Figures I.E.9-12
A P-CURVE DEFINES A MOVEMENT

selections of cels from cel classes that have been made in that frame. Specifically, the system first displays light buttons (stylus targets) listing all cel classes. Whenever the animator points to a particular cel then visible in the frame, it is replaced by the next cel in the class. When he points to the last cel, it is replaced by the zeroth cel, that is the empty or invisible cel. When he now points to the name of the class, its first cel is again selected and displayed. The light buttons also serve another function. Pointing to a particular button "connects" two knobs to that cel class, such that turning the knobs will readjust the position of the cels belonging to that class (all the cels, not just the one currently visible). Further details are given in Figures I.E.13-14.

Miscellaneous commands exist to restart the system, to exit from the system, to name and save an animation sequence, that is a complex of cel classes and movement descriptions, to recall for further work an existent sequence, and to play back the current sequence. There are modes which determine whether straight lines are drawn solid or dotted, and whether a grid is activated or not, and instructions for establishing the density of the grid. There are commands to display all members of all cel classes belonging to a particular movie, to display the movie in frame j while cel k of class 'abcd' is under construction, to display the next or the previous frame, to show or hide the picture of the para-



(13)



(14)

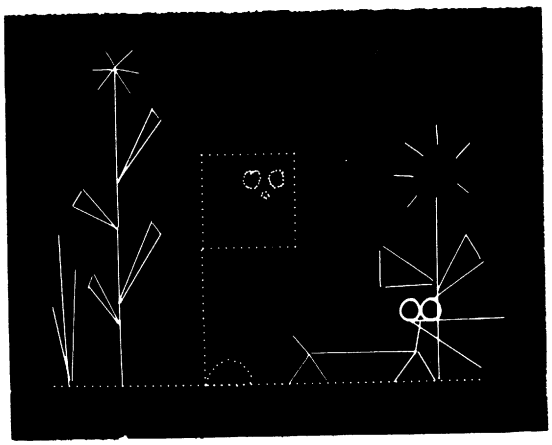
(13) All cels used in the animation of Oopy--he flaps his ear, winks, and sticks out his tongue--are shown superimposed. These sketches were also done by Mrs. Johnson.

(14) GENESYS is in frame-mode. The current state of a particular frame is displayed. Also visible are "light-buttons" representing cel classes (mouth, tongue, eye, ear, brow). The animator may alter the current frame, switching the selection of a cel from a class by pointing at it, or changing its position by turning knobs located under the scope. The underlying movement descriptions are automatically updated by GENESYS.

Figures I.E.13-14
CELS AND CEL CLASSES

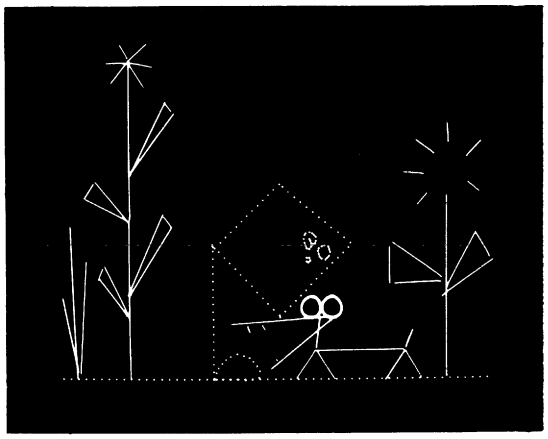
metric curve of a cel class, and to call the editing system with which one refines movement descriptions. (*4)

The system has been used in the construction of short cartoon and abstract dynamic sequences by several individuals with varying degrees of artistic skill and training in animation. One cartoon with a particularly interesting evolution is described in Figures I.E.15-18. All users were delighted with the responsiveness of the medium, with the immediate display of their animation. All were discouraged by the unreliability of the system. Despite numerous system crashes and program bugs, sequence of 5 to 10 seconds duration were composed, often with little advance preparation, in 1/2 to several hours of console time. Except for that shown in Figures I.E.15-18, all these sequences were exceedingly simple, and usually involved no interaction between objects. Undoubtedly, animation with GENESYS would be much faster if the system were engineered more smoothly, more completely debugged, and then stabilized, if the animator were accustomed to working with it, and if he were working on a longer sequence where the same cels could be used again and again.



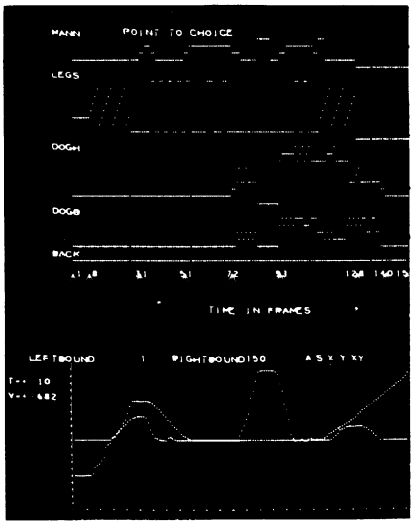
A man, tripping blithely along, kicks a dog lying in his path. The dog rises and trots off to the right (shown above). It then returns, teeth bared (shown in Figure I.E.16), and bites the man. The man jumps and runs away. The dog first follows, then returns once again to rest. The duration of the sequence is approximately 20 seconds. (Please turn to Figure I.E.16.)

Figure I.E.15
A SHORT CARTOON--WHAT THE VIEWER SEES

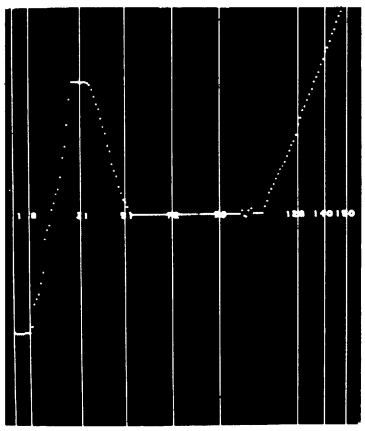


Mr. Ephraim Cohen of Orange, New Jersey, a mathematician and programmer who is also a skilled caricaturist, completed the cels for his cartoon one week-end afternoon at the TX-2. The system then crashed, and he was forced to return home. He sent me through the mail four selection descriptions, to choose cels from the classes "man's head", "man's legs", "dog's head", and "dog's body", and two path descriptions, to drive horizontally the man and the dog. I input the dynamic descriptions, viewed the result, and then refined the movie by several iterations of editing the descriptions and viewing the sequence. (Please turn to Figure I.E.17.)

Figure I.E.16
A SHORT CARTOON--HOW IT WAS MADE



(17)



(18)

(17) The dynamic descriptions defining Mr. Cohen's cartoon as of January, 1969, are shown above. The selection descriptions, from top to bottom, belong to the man's head, the man's legs, the dog's head, and the dog's body. There are 4, 8, 8, and 4 cels in each class, respectively. The two waveforms represent the changes with time of the horizontal coordinates of the man and the dog.

(18) The man's waveform is shown superimposed on rhythm description representing critical instants of time. This display was used in the process of resketching the waveform to refine the dynamics of the film.

Figures I.E.17-18
A SHORT CARTOON--WHY IT WORKS

FOOTNOTES -- I.E.

(*1) A discussion of the practical problems involved in achieving real-time computation and playback of animation sequences appears in III.B.

(*2) Mermelstein and Eden,[64] p. 268.

(*3) Halas and Manvell,[3] pp. 61-68. They explain further:

> ". . .
>
> "Exaggeration in cartoon is largely born of function. The gesturing parts of the body --legs, arms, hands, feet--demand enlargement, and above all so does the head, as the principal feature in either human or animal kind. Within the head itself, the mouth demands enlargement through the gesture of speech, and so does the eye, because it is the most important means whereby mood, emotion and character may visually be demonstrated.
>
> "Finally, there is a technical reason in cartoon film-making for exaggerations and simplifications. When drawings have to be produced in large numbers and by various hands, some form of graphic economy through the emphasis of certain characteristics is essential to enable the nature of a figure to be maintained without any unwanted deviations which would be likely to confuse the clear markings of character.
>
> ". . .
>
> "When inanimate objects assume life in the pursuit of humour, the need for exaggeration becomes even more pronounced. For example, the cartoonist may want to give an automobile the characteristics of a dog in its attitude to the fuel that its owner offers it. When it shakes its shaggy head in refusal to tank up with the wrong brand of spirit, then the whole body of the car must shake like a neurotic Saint-Bernard, and its head-lamps must become great eyes filled with a pathetic dismay.
>
> "In instructional films where comic exaggeration is needed for some emphasis of fact, even diagrammatic models may be permitted to acquire human or animal personalities, though exaggeration of this kind is only used to help make some important teaching point.

"Visual symbols may have to be introduced to emphasize what cannot in real life be seen but only felt. The wind may enter the action in the form of the moving lines familiar in the strip cartoon; hot and cold may also require some kind of symbolic indication made by lines which surround a figure in order to indicate shivering or sweating. The exaggeration here lies not only in the experiences shown through the reaction of the characters, but in the way the graphic symbols are introduced and animated.

". . .

"The basic aesthetic principle in cartoons of graphic distortion is supplemented in entertainment cartoons by the fun that can be had at the expense of the physical laws of motion. The exploitation of actions which go beyond all normal physical possibilities in the real world have a powerful effect on any audience.

". . .

"Time is also a plaything in the cartoon--an action can be speeded to such a degree that round the world in eighty days becomes round the world in as many seconds. The chase motif of movie drama develops into a paroxysm of speed with a fine graphic flourish in perspective--the minute dots on the horizon hurtle with rhythmical abandon into vast blown-up proportions in the foreground all within three or four seconds. The cartoon uses slow motion far less than quick because its peculiar dynamic usually makes it move faster than life in everything it does.

"For instructional animation, however, a slow speed is often desirable; the stripping down of a complex diagram to its inner sections may well require a slower-than-life rhythm--that is, were the sections of real machines being lifted apart for inspection. The diagram unpeels in synchronization with the commentary to meet the demands of clear and well-timed exposition.

". . ."

(*4) Seeing frame j while constructing cel k of the class 'leg' facilitates drawing a position of the leg so that it kicks the soccer ball in that frame. This illustrates one feature of GENESYS designed to combat the problem, discussed extensively in I.C.7. and I.C.8., of relating the simultaneous motions of several objects.

I.F. CONCLUSION -- THE REPRESENTATION OF DYNAMIC INFORMATION -- THE CONCEPT OF A DYNAMIC DISPLAY

Thus the essence of picture-driven animation is:

(1) That one may formally isolate aspects of picture change which recur over extended intervals of time, and define picture-transforming algorithms which produce this change;

(2) That there exists a set of abstractions of dynamic information, data sequences which drive these algorithms to produce a large and interesting class of animated displays; and,

(3) That these abstractions may in turn be modeled, generated, and modified by static as well as animated pictures, modeled in the sense that the picture structure represents the data sequence, generated and modified in the sense that the picture represents the process of synthesis as well.

The three kinds of dynamic descriptions constitute a rich, expressive, intuitively meaningful vocabulary for dynamics. Each type abstracts an important category of dynamic behavior--flow and continuous change (path descriptions), switching and repetitive choice (selection descriptions), and rhythm and synchrony (rhythm descriptions). The vocabulary is economical, flexible, and general in the

sense that it can characterize the dynamic similarities that exist in seemingly diverse animation sequences.

The use of dynamic descriptions couples picture definition by sketching and by algorithm; it furthermore allows both local (of the individual frame) and global (for an interval of time) control over dynamics. We have chosen to stress the latter and adopted the term "global dynamic description," for it is the capacity for global control that results uniquely from the use of the computer as an animation medium. Yet a dynamic description is not only a representation over an interval, but a sequence of single elements whose modification also provides local control over individual frames. Both local and global control are vital to the successful synthesis of movement. He who accidentally crashes into a wall while running from the police is going from the continuous to the discrete, from a global motion to a local event. He who aims to scale the wall is interpolating the continuous between the discrete, adjusting the global to fit the constraints of the local. We have seen how GENESYS facilitates global control (Figures I.E.8-12) as well as local control (Figure I.E.14).

The naturalness and power of the vocabulary is increased by the ability to manipulate it in an interactive graphics environment. There exist, for each kind of data sequence, static pictorial representations such as the waveform and the parametric curve which provide a global view of and

facilitate precision control of the temporal behavior implied by the sequences. There exist, for each kind of data sequence, methods of dynamic specification such as the clocked sketching of a parametric curve and the tapping of a rhythm which allow the animator's sense of time to be transmitted directly through the medium of the computer into the animated display. We use the term "global dynamic description" and the names of the three types somewhat loosely in referring both to the underlying dynamic data sequences and to their corresponding pictorial representations. The imprecision is purposeful, for it is very significant that, in an interactive graphics environment, one can easily traverse in either direction any leg of the triangle {Dynamic Data Sequence, Static Pictorial Representation, Dynamic Pictorial Representation}. What results is an important plasticity in the representation of dynamics. Characterizations of change can be manipulated (shifted, stretched, superimposed, . . .) within and between the domains of the static and the dynamic. Several animation sequences can readily be related, coordinated, or unified, regardless of whether or not they ever occur concurrently. Dynamic behavior (data) can readily be transferred from one animation subsequence (including the animator) to another, from one mode of representation or embodiment in a picture to another.

Our concept of a dynamic display is a broad one, and purposely so. For as we stress in Part II, a computer-

mediated display is not only what is visible but what is contained in its model in the computer system. And the system, i.e., an interactive animation system, includes not only disks and core but an animator and perhaps an ongoing physics experiment as well as a tape-recorded speech. This system evolves continually through real time. Occasionally there occurs a particular reorganization of the system which results in the transfer of information from the animator to the pictorial data base, or in a computation on the data base which results in a sequence of visual images, that is, data directly convertible by hardware into visual images. Thus, as we have stressed before, the act of mimicking dynamics is a (user-driven) dynamic display. This unification of the roles of picture and action, as equivalent components of displays, is important. A flexible interactive computer-mediated animation system must model the animator and his actions (input), dynamic data (storage), and static and dynamic pictures (output) as interchangeable representations of movement and rhythm, all mediated by the computer through its store of algorithms to produce animated system behavior.

The greater is the number and generality of available models of pictures and of processes of picture construction (actions), the more flexible and powerful is the animation system in its ability to deal with dynamic information. The previous chapters summarize our experience with special-purpose animation systems that present the user with a fixed set of

such models. The multi-purpose language of Part II allows the animator himself to synthesize new models.

With such a language one can describe arbitrary action-picture interpreters that extract dynamic descriptions from the animator's use of system devices and transform them and existing static and dynamic displays into new static and dynamic displays. In Part I, particularly in I.C., we have seen some simple yet useful examples. Two others follow:

(1) Another recent innovation at Computer Image Corporation[29] is a technique that converts a speech waveform into the appropriate selection description to drive a pictorial model for the mouths of cartoon characters. The data sequence selects from among seven standard mouth shapes and positions, which may be found in typical animation handbooks such as Halas and Manvell.[3] Some observers have claimed that this is done so well that one can identify the language that is being spoken by observing only the visible movements of the mouth. Their interpreter which carries out this task has been implemented in hardware, but the goal, transforming dynamic information from one representation to another, is the same.

(2) A second potential application is multi-animator animation. A program could interpret the actions from various sets of devices (styli, togs, etc) and the pictures on numerous scopes. Several animators could mimic interesting roles, aiding the achievement of the desired pacing, give-

and-take, and synchrony required for certain classes of films.

Finally, the use of dynamic descriptions helps establish a conceptual framework which facilitates efficient use of the resources of the animation system: animator, software, and hardware. The process of animation may easily be factored into the synthesis of various dynamic strands. We shall see in III.B. that a sequence will be constructed in multiple passes in which the computation of some strands can aid the later synthesis of other strands. The representation of dynamics by pictures has a valuable by-product --all system tools for the construction of animation images may be made applicable to the specification of dynamic information. And, lastly, we shall present, in Chapter III.B., hardware additions suggested by the utility of movement descriptions, additions which when incorporated in an animation scope should reduce the overall cost of computer-mediated animation.

We have seen that there are advantages and disadvantages to each of several approaches to the definition of dynamic pictures--the construction of individual frames, the algorithmic generation of sequences of frames, and picture-driven animation. This suggests that a flexible animation system would allow the harmonious blending of all these techniques. Here GENESYS fails a priori, for it makes inaccessible the full computational power of the computer. Within the language of GENESYS one cannot implement algorithms by writing programs. We have also seen that the GENESYS is inadequate because it presents the animator with a fixed set of commands and tools, with fixed mechanisms of control, and with fixed models of pictures and of processes of picture construction. A suitably skilled animator may himself determine these aspects of his animation system, his animation-machine, only if the system is not a fixed set of commands but an extensible, truly open-ended programming language.

Strong constraints need be applied to such a language. Its domain of discourse must include algebraic computations, the manipulation of structured data and pictures, the generation of displays, and the monitoring of user interaction. Yet, unlike most programming languages, it must develop these constructs in a clear and simple manner that is harmonious with a layman's intuition about pictures and the construction of pictures. (By laymen we mean animators,

artists, educators, not programmers.) The design of a language which seeks to meet these goals is outlined in Part II of the dissertation.

II.A. A GENERAL INTRODUCTION TO THE DESIGN OF AN OPEN-ENDED, MULTI-PURPOSE, INTERACTIVE COMPUTER-MEDIATED ANIMATION SYSTEM

II.A. A GENERAL INTRODUCTION TO THE DESIGN OF AN [illegible]

II.A.1. THE ESSENTIAL FEATURES OF SKETCHPAD, BEFLIX, AND CAFE

The purpose of this chapter is further to motivate aspects of the design of an open-ended, multi-purpose interactive computer-mediated animation system. The system components that enable interactive computer-mediated animation were detailed in Chapter I.A. and are restated briefly here:

(1) a general-purpose digital computer;

(2) a hierarchy of auxiliary storage;

(3) an input device which allows direct, real-time, drawing to the computer in at least two spatial dimensions;

(4) an output device which allows direct,real-time, viewing of animated displays;

(5) a "language" for the construction and manipulation of static pictures;

(6) a "language" for the representation and definition of picture change and the dynamics of picture change;

(7) mechanisms for converting the specifications of static picture structure and picture dynamics into a sequence of visual images; and,

(8) mechanisms for storing, retrieving, and displaying this sequence of visual images.

Yet such a description provides only a general foundation upon which specific designs can be based. Several concrete realizations, special-purpose animation systems, were described in Chapter I.E. Guidelines for a more general and comprehensive approach emerge from experience with these systems and from the experience of others. We shall therefore

discuss in some detail three particularly interesting and relevant pieces of work: SKETCHPAD (1963), the first systematic and comprehensive system for on-line graphical man-machine communication;[39] BEFLIX (1964), the first programming language for computer animation;[5,6] and CAFE (1968), an on-line typewriter-controlled animation command language.[26] The The design of APPL, a new Animation and Picture Processing Language, draws heavily upon observations about SKETCHPAD, BEFLIX, CAFE, and GENESYS.

SKETCHPAD is significant because it presented numerous techniques for man-machine graphical communication which were based upon some powerful new abstractions. SKETCHPAD demonstrated that the computer could be an active partner in the processes of sketching pictures and defining their structure. A command language of demonstrative light-pen movements and push-button signals is used to control the construction and selective deletion, replication, and modification of picture parts. Geometrical relationships can be imposed implicitly through the application of constraints, such as the parallelism of lines or the colinearity of points. Thereby SKETCHPAD can turn certain sloppy sketches into precise ones. SKETCHPAD stores explicit information about the topology or structure of pictures. This facilitates the specification of complex commands through the recursive application of a simple command (such as "move") on the various elements of a hierarchic picture structure.

Economy in the effort of picture construction is aided by repeated use of existent pictures and structures. The copy function duplicates pictures and constraints and leaves the copies free for arbitrary modifications. The instance function also duplicates pictures, however allows the copy to differ from the master in position, orientation, and scale only, and preserves the link between them such that changes in the master are reflected in changes in its instances.

Yet in his ability to model, assemble and disassemble, and name picture structures, to undo the effect of previous operations, and to extend the effective command power of the system, the SKETCHPAD user is quite limited. Four sources of these limitations are now suggested:

(1) SKETCHPAD is not a programming language. It is a command language, in which each command is executed as soon as it is invoked, and thus at each step the user determines the flow of control from command to command. The exception is in the application of constraints, in which SKETCHPAD simulates a descriptive (non-procedural) language capable of solving certain implicit specifications of picture geometry. However, the user cannot command SKETCHPAD to compute an arbitrary function on pictures.

(2) SKETCHPAD is a fixed system of commands, a closed system, lacking the capacity for definitional extension of the command structure, or the hierarchic definition of new concepts in terms of existent ones. Only sets of data items,

picture and constraint complexes, may be coalesced and used as units. The instance simulates a procedure-writing capability that is restricted to programs that generate fixed-geometry subpictures, and whose four parameters are the horizontal and vertical positions, the orientation, and the scale. Instances cannot simulate the effect of procedures that generate variable-geometry or-topology pictures. Yet instances are the only structures that may be embedded hierarchicly to arbitrary depth.

(3) The allowable topology or structuring of pictures is built into the system and is often not general enough or powerful enough. Relationships between arbitrary picture elements can only be established by including them in the same master picture, yet an element can only be a member of one master. The characterization of pictures is too dependent upon what is required for their immediate display. The ability to change a picture is too dependent upon the structure imposed during picture generation. It may therefore become difficult to undo or alter what one has done. I.E. Sutherland noted that "... there is a definite need for a general-purpose function for making topological changes to a drawing...."[39] What is required is a richer user language for establishing and manipulating picture structure.

(4) Finally, the ability to name and retrieve picture parts is limited in SKETCHPAD to pointing and to the designation of numbers assigned in the course of picture construction.

There is no way to isolate subpictures by a citation of their identifying properties, by an implicit description such as 'all those picture parts to the left of the rectangle and inside the circle.' One source of rigidity in making picture changes is the difficulty of naming through enumeration all the subpictures involved in the change.

BEFLIX's significance stems from its pioneering of the use of the computer in animation and its existence as the first programming language for the specification of picture dynamics. (*1) A written language embedded in MACRO-FAP, BEFLIX constitutes an open-ended system which may be extended by the user. The animator constructs a picture by symbolically paointing intensities on rectangular elements of one of a number of finely-divided surfaces or rasters, and directing an imaginary camera at a particular surface. This causes a picture of the surface to be recorded on microfilm. After the picture is modified to form the next frame, the camera again records its state. A movie can be made by repeating this cycle.

The language consists of a "scanner language" and a "movie language." With the scanner language, the user manipulates bugs or scanners over the picture raster, performing tests on the positions and the intensities of the covered elements. The results of these tests determine subsequent changes to bug positions and element values. With the movie language, most of which was implemented in the scanner language,

one controls the output or temporary storage of pictures, performs static or dynamic drafting operations, and modifies the contents of rectangular areas. Dynamic operations are interrupted during the output of intermediate frames. The operations upon rectangular areas provide powerful control over texture, a feature exploited to advantage in the BEFLIX films of Vanderbeeck.[21]

CAFE is a command language for the on-line construction, editing, and display of figures consisting of collections of straight line segments. The work, like ours, has been carried out within the past two years at the M.I.T. Lincoln Laboratory. "On-line" in CAFE means typewriter communication in a time-shared environment. Static pictures are constructed by invoking commands to assign coordinates to points and lines, and then group them into "objects" and further into composites called "graphs." Graphs are further grouped into dynamic units called "elements," each of which has its own independent motion description. A motion description is defined by specifying a sequence of "scenes." Each scene has its own time interval, and consists of a sequence of translations, rotations, and scale changes. The motion of elements may be related by establishing a hierarchic dependence, causing one element to follow another (add the second element's motion to its own). Picture and scene descriptions are kept symbolically for further modification. Output is either to a microfilm recorder or to a scope for direct viewing, but the environment

is too heavily loaded to allow [illegible] playback

of a movie.

Several comparisons of BBFLIX, CAFE, and GENESYS are relevant to the design of a new computer animation system:

(1) BBFLIX's off-line operation and CAFE's on-line typewriter control contrast sharply with the direct graphical communication of SKETCHPAD and GENESYS. (*2) Ways to exploit an interactive environment in animation have been developed in Part I of the dissertation; Part II explains how ASPEN's design facilitates the flexible application of these concepts.

(2) BBFLIX and CAFE differ significantly in the power of the resulting animation language. BBFLIX is a complete programming language in which one can in principle express any computable function. Macro-definitions, which may be nested to arbitrary depth, allow definitional extension of the command structure of the language. CAFE and GENESYS are command languages, incapable of expressing arbitrary computable functions. Like any good on-line system, CAFE allows the saving, copying, and modification of files describing particular static pictures and the scenes that generate dynamics; GENESYS allows the saving, copying, and modification of pictures and dynamic data sequences. Maintenance of CAFE files, which could simulate nested calls of picture-defining subroutines, cannot be established. Thus, as in GENESYS, definitional extension is impossible; the command structure is fixed.

II.A.2. SOME SPECIFIC ISSUES RAISED BY BEFLIX, CAFE, AND GENESYS

Several comparisons of BEFLIX, CAFE, and GENESYS are relevant to the design of a new computer animation system:

(1) BEFLIX's off-line operation and CAFE's on-line typewriter control contrast sharply with the direct graphical communication of SKETCHPAD and GENESYS.(*2) Ways to exploit an interactive environment in animation have been developed in Part I of the dissertation; Part II explains how APPL's design facilitates the flexible application of these concepts.

(2) BEFLIX and CAFE differ significantly in the power of the resulting animation language. BEFLIX is a complete programming language in which one can in principle express any computable function. Macro-definitions, which may be nested to arbitrary depth, allow definitional extension of the command structure of the language. CAFE and GENESYS are command languages, incapable of expressing arbitrary computable functions. Like any good on-line system, CAFE allows the saving, copying, and modification of files describing particular static pictures and the scenes that generate dynamics; GENESYS allows the saving, copying, and modification of pictures and dynamic data sequences. Hierarchies of CAFE files, which could simulate nested calls of picture-defining subroutines, cannot be established. Thus, as in GENESYS, definitional extension is impossible; the command structure is fixed.

(3) BEFLIX, CAFE, and GENESYS differ in the mechanisms of control they provide over the temporal dimension. The BEFLIX animator specifies an algorithm that determines the picture at successive instants of time. The CAFE animator defines an ordered succession of scenes, specifying the duration of each one and the set of transformations to be applied during that interval. The GENESYS animator defines, through the sketching, mimicking, and graphical editing of pictures, data sequences that control movement and rhythm in an animation sequence. Only in BEFLIX can the lengths of phases of the activity or the nature of the constituent picture change be made to depend upon a computation of some aspect of the sequence; in CAFE and GENESYS this must all be expressed explicitly in advance.

(4) Two kinds of picture change occur in the process of animation; one yields successive stages in the construction of a static picture, the other produces the actual dynamics to be recorded on film. In BEFLIX one language is used for both tasks. The effect of a sequence of statements that alter a picture depends upon whether or not the instructions are accompanied by "camera" calls, whose result is the output of frames to the microfilm recorder and the advance of movie time. If camera calls are included, a dynamic sequence is formed; if there are no camera calls, a new static image is formed. In CAFE and in GENESYS, on the other hand, there is a strict separation between static picture definition and

the imposition of dynamics. Thus there are in CAFE two operations which translate pictures, a shift command (static) and a move command (dynamic).

(5) Finally, there is the significant issue of parallelism or the description of concurrent dynamic activity. BEFLIX is a language with strictly sequential or explicit flow of control, including transfer of control through standard mechanisms of subroutine calls. Camera calls must be moved within the program if additional concurrent picture changes are added. CAFE and GENESYS allow the independent definition of concurrent strands of dynamic activity, parallel actions in a scene, allowing new strands to be specified without altering existent ones.

Our conclusions are that we desire: (1) direct graphical interaction; (2) an extensible programming language; (3) flexible control over the dimension of movie time; (4) unique mechanisms for producing picture change; and, (5) the independent definition of concurrent strands of dynamic activity. These and other criteria in the design of APPL are expanded in the next section.

II.A.3. ESSENTIAL FEATURES IN THE DESIGN OF APPL

This background has motivated the following aspects of the design of APPL, a new Animation and Picture Processing Language:

(1) Like any interactive computer graphics system, APPL contains commands which are immediately executed when designated. These commands can also be grouped into procedures whose execution is delayed until the name of the procedure is activated.

(2) APPL is capable of definitional extension, through the nesting of these procedures to arbitrary depth. To facilitate the nested and recursive application of procedures, they are allowed to take arbitrary numbers of parameters, and reasonable conventions regarding the scope of variables and the sharing of procedure data are adopted.

(3) APPL is a complete programming language, in which it is possible to express any computable function, as well as powerful animation and picture processing operations. It attempts to achieve these goals in the simplest, most direct, and most general way possible. Thus, included in the base system are only one distinct numerical data type, the SCALAR, and one distinct pictorial data type, the POINT, augmented by powerful mechanisms for aggregating these units and for defining complex picture types in terms of simple ones.

These three criteria guarantee that the system is open-ended in such a way that significant extensions to the

command structure may be expressed within the system. The continuity between immediate execution mode and procedure definition mode, characteristic of conversational, JOSS-like languages, should aid the system utilization of those with little or no programming experience, for portions of potential programs can easily be tested as they are devised. (*3)

(4) I.A.2. proposed that a single set of APPL commands produce the picture change that sometimes yields successive stages in the construction of a static picture, and that other times yields successive frames to be included in the movie itself. These commands are called PICTORIAL OPERATIONS. They cause the introduction, replication, and deletion of pictures, and the modification of both their directly visible properties, such as location and intensity, and the structure of their internal representations such as picture-subpicture relationships.

(5) Picture changes that result in dynamic sequences must be organized with respect to a scale of movie time or simulated time. APPL contains mechanisms to trigger and control the passage of movie time. Because the user may interrupt the passage of movie time, and may start and stop it at will, these mechanisms are more complex than the corresponding ones in BEFLIX.

(6) APPL provides for the definition and use of global dynamic descriptions within the framework established for

controlling the passage of movie time. The desired goal is the flexibility and plasticity in representing and using dynamic information that was discussed in I.F. Therefore, commands that construct and manipulate pictures of dynamic descriptions are implemented via definitional extension of the language, so that their meaning can easily be changed by the user.

(7) The animator is as an integral component of animated system behavior. APPL includes constructs for monitoring the animator's use of the stylus, toggle switches, push buttons, and other devices. This aspect of the language design must be clean and simple, for it will be used to construct the control features of interactive graphical procedures. If large parts of an animation system are their implemented via definitional extension, the animator can easily control his style of interaction with the system. (*4)

(8) APPL allows quasi-parallel flow of control during the passage of movie time. This facilitates the independent specification of the parallel dynamic strands representing concurrent activities of several objects in one movie. Mechanisms exist for communication between programs implementing these activities. We distinguish between the master copy of a program and its use, or activation as part of an animation sequence. Several instances of one program may

run concurrently through an interval of movie time.

(9) The ability to impose structure on pictures is provided by the language. A data structure called the AGGREGATE facilitates the modeling of global dynamic descriptions and of complex sequences and hierarchies of pictures. The APPL user can define new picture types as structured composities of existing picture types; recursive definitions are allowed.

(10) APPL allows the user to talk about the characteristics of pictures. Pictures are characterized by a set of ATTRIBUTES, including 'type', taking on values called PROPERTIES. The user can enlarge the set of picture attributed by definitional extension of the language. When a picture possesses certain properties, this will be said to define a picture state. Tests for the existence of picture states, called pictorial conditions, will control the evolution of an animation sequence.

The last two criteria guarantee that a rich picture description capability may be implemented via definitional extension of APPL. There may be developed a coherent family of animation languages, all grown from the common base, or core language, each uniquely suited to the manipulation of a particular kind of picture, and each capable of being made still richer through further extension. (*5)

II.A.4. A SCENARIO ILLUSTRATING SOME USES OF APPL

To illustrate the process of animation in an interactive programming environment, we present a scenario. The example could be executed with a system that is an extension of the APPL base language described in the following chapters. Specifically, we assume that the base language has been extended so that it provides a capability for picture-driven animation such as exists in GENESYS. What follows is intended as a general picture of how APPL may be applied, and therefore does not indicate all of the specific mechanisms necessary to create the effects described.

The animator, having completed 'SPROINGBOINGZAP' as described in I.A.3., desires to experiment further with the motif of a moving triangle. Specifically, he intends to construct various static patterns and textures composed of triangular parts, and then use these as backgrounds and masks against which and through which a few dynamic triangles dance and cavort. Therefore, he will extend APPL to facilitate the construction of triangles and patterns formed of triangles, and the definition of both isolated and coordinated movements and deformations of collections of triangles. (*6)

How it is done:

Step 1--The construction of a single triangle, first by the animator, then by the system:

USER(U): *CALL* APPL;

APPL(A): *HELLO, THIS IS APPL;*

(APPL is entered at the "command mode" level. Designating a command while the system is in this mode causes its execution to begin immediately.)

U: *NEWPT* PT.A *at ... ;*

(After activating the command *NEWPT*, and entering a name for the point, 'PT.A', the animator touches the stylus to the location whose x coordinate is 0.0 and whose y coordinate is 0.2. (*7) A point is created at that location.)

U: *NEWPT* PT.B *at ... ;*

U: *NEWPT* PT.C *at ... ;*

U: *CONNECT* PT.A *to* PT.B;

(After activating the command *CONNECT*, he points at the two points named PT.A and PT.B. Since APPL expects points as arguments for the command, these have become active targets for the stylus. A line then appears between PT.A and PT.B.)

U: *CONNECT* PT.B *to* PT.C;

U: *CONNECT* PT.C *to* PT.A;

(Construction of the triangle is now complete.)

.....

(Rather than carry out this sequence of steps whenever a triangle is to be constructed, the animator will extend APPL so that it aids future construction.)

.....

(He invokes the system meta-command *DEFINE*, and enters 'BUILDATRIANGLE' as the name of the command to be defined. This puts the system in "definition mode." When a legal command is now designated, it is not immediately executed, but rather placed in an ordered, numbered, list of commands whose execution is delayed until later activation under the command name *BUILDATRIANGLE*.)

U:

PROGRAM #1

```
DEFINE BUILDATRIANGLE;
.01     NEWPT PT.A at 0.0,0.2;
.02     NEWPT PT.B at 0.1,0.0;
.03     NEWPT PT.C at -0.1,0.0;
.04     CONNECT PT.A to PT.B;
.05     CONNECT PT.B to PT.C;
.06     CONNECT PT.C to PT.A;
END;
```

(The definition, whose conventions are explained in more detail in II.B.2., is terminated by the system meta-command *END*. Now there is a new APPL command; the animator checks its meaning by activating it.)

U: *BUILDATRIANGLE;*

(A triangle, identical to the one constructed before, appears in the working area of the scope. Thus, APPL has been extended to include a triangle-construction command.)

Step 2--The construction of two APPL programs which aid the interactive generation of more triangles:

(*BUILDATRIANGLE* is useless, for it always constructs the very same triangle. Hence the program is now made a paradigm for defining two new commands which will aid the generation of many triangles.)

U:

```
DEFINE FORMTRIANGLE among PT.1 and PT.2 and PT.3;
.01     CONNECT PT.1 to PT.2;
.02     CONNECT PT.2 to PT.3;
.03     CONNECT PT.3 to PT.1;
END;
```

PROGRAM #2

.....

(Suppose that PT.D, PT.E, and PT.F are arbitrary points visible on the scope. To check the definition of the new command, the animator selects these three points with the stylus, thereby designating them as parameters for *FORMTRIANGLE*.)

U: *FORMTRIANGLE among* PT.D *and* PT.E *and* PT.F;

(The three points are automatically connected to form a triangle.)

.....

U:

```
DEFINE DRAWTRIANGLE;
.01     NEWPT PT.4 at STYLUS;
.02     NEWPT PT.5 at STYLUS;
.03     NEWPT PT.6 at STYLUS;
.04     FORMTRIANGLE among PT.4 and PT.5 and PT.6;
END;
```

PROGRAM #3

(This program differs from the others in two ways. First, it includes a command itself defined by another program, thereby demonstrating APPL's open-ended character noted in II.A.3. Once defined by language extension, a command is indistinguishable from those included in the base language, except that its meaning can easily be changed.

Second, the program includes a command whose argument must be supplied by the user during execution. Before generating a point, the system waits for the stylus to be positioned on the tablet surface, just as it waits for an argument when the *NEWPT* command is directly activated by the user.)

.....

U: *DRAWTRIANGLE;*

(After activating this new command, the user positions the stylus three times at arbitrary locations on the surface; points appear at these locations, and automatically become the vertices of a triangle.)

Step 3--<u>The construction of dynamic triangles using picture-driven animation, followed by augmentation of the basic algorithm of picture-driven animation</u>:

The animator experiments with moving triangles, using the APPL facilities for picture-driven animation. Suppose that, as in GENESYS, the system currently allows continuous translation of cels, but not continuous scaling. In contrast to GENESYS, however, the problem can here be solved smoothly within the system. We recall that the basic algorithm

implementing picture-driven animation is one which transforms cels, and selects among them, according to successive values of a set of data sequences called global dynamic descriptions. In APPL, this algorithm is expressed as a simple APPL program, and hence is easily modified on-line. (Recall Features 5 and 6, in the previous section.) Modifications might be made, for example, so that cels can be transformed continuously through scaling and rotation, being driven by path descriptions, and discretely through reflection about the horizontal axis, being driven by a selection description which determines whether the triangle is pointed 'up' or 'down'.

Step 4--The algorithmic definition of a particular movement:

Tools for the free-hand sketching of dynamic descriptions are implemented in APPL through language extension. The animator now wants to see the effect of a regular variation, such as a sawtooth function, when it is applied to the scale factor of a triangle. Thus he writes in APPL a routine which computes this path description.

Step 5--Additions and modifications to the subsystem for defining and refining global dynamic descriptions:

In APPL, unlike GENESYS, the user has full control over the interpretation of the motion of the stylus, since he can write programs which monitor its behavior. (Recall Feature 7 in the previous section.) A library of routines for constructing and editing pictorial representations of dynamic

descriptions may therefore be written in APPL. The animator can then take an existing program which accepts p-curves and modify it, for example, so that the resulting path description is defined as the variation of the radial coordinate rather than that of the horizontal coordinate. (*8) He can also tailor to his own specifications such conventions as the signals he gives the system when he wants to start and terminate the real time sketch.

Step 6--The construction of a subsystem for the generation of complex static patterns of triangles:

As the animator works, he calls again and again, with varying parameters, the same triangle-generating and triangle-modifying commands. Often he activates a function with only one of its arguments varied from the previous call, but must awkwardly specify its complete argument list again. He also designates certain command sequences with regularity, but must awkwardly repeat the same actions each time. He therefore writes a small control program in the animation language, one that automates many of the time-consuming aspects of re-designation of parameters and command sequences, and defines the flow of control among commands. Parameters are stored in the control program's local data base; commonly executed command sequences are made subroutines of the program. Use of this language extension speeds the search for interesting patterns, which are stored in the system-wide (global) data base when the control program's execution is terminated. (*9)

Step 7--Extension of APPL to facilitate operations over triangular patterns:

Operations over complex patterns are facilitated by a language with which one can name, describe, and characterize subpatterns. The animator establishes a standard data structure with which to model triangles and defines a set of attributes which characterize them. (Recall Features 9 and 10 in the previous section.) These definitions allow him to communicate with APPL about a pattern in terms of such concepts as its center, its area, the number of triangular subpatterns, and its state of geometrical inclusion or exclusion with respect to other patterns. The animator may then refer implicitly to a set of triangles as "all those enclosed by triangle A," and need not always resort to naming explicitly or pointing to each one. (Recall the fourth limitation of SKETCHPAD listed in II.A.1.) This is particularly useful if the set of triangles varies in a complex manner throughout a film. As APPL becomes richer through the addition of new picture structures and attributes, it becomes possible flexibly to construct, retrieve, and manipulate complex pattern configurations.

Step 8--Construction and activation of APPL programs that generate complex dynamic patterns of triangles:

Some of the desired dynamic effects are achieved with cels, dynamic descriptions, and the picture-driven animation algorithm. Others, where the eventual evolution of an animation

subsequence depends in some complicated manner upon earlier aspects of the sequence, are defined directly by new APPL programs. This is done by using the APPL constructs that control the advance of movie time. Given a number of such programs which compute parallel actions in a movie, any or all of them can be activated for quasi-parallel execution, which is synchronized automatically by APPL. (Recall Feature 8 in the previous section.) Thus the animator controls which aspects of the movie are computed and/or displayed concurrently. Computation or playback may be interrupted at any time to allow redefinition of both pictures and programs. The ability to decompose the construction of a sequence, to synthesize it piece-by-piece, in modular fashion, facilitates effective real-time interaction by reducing the short term computation and display demands on the system.

II.A.5. IMPLICATIONS OF THE SCENARIO

The example illustrates the diversity of possible uses of APPL, including:

(1) The mediation of direct user requests to construct or modify pictures, such as the first triangle in Step 1; the algorithmic definition of dynamic displays, as is done directly in Step 8, and is done in part by:

(a) The algorithmic definition of static pictures, such as the second triangle in Step 1;

(b) The algorithmic definition of picture dynamics expressed by movement and rhythm descriptions, such as the sawtooth function in Step 4; and,

(c) After significant extensions to enrich the descriptive power of the language, the manipulation of a particular class of structured pictures, such as the triangular patterns in Step 7. A version of APPL that has been augmented in this way is one of the coherent family of animation languages that was mentioned at the end of II.A.3. In the example at hand, APPL is made into a language well-suited for the animation of triangles and triangular patterns.

(2) The building of a set of tools for interactive computer-mediated animation, such as:

(a) A set of programs that aid the construction of

static pictures, such as the triangles of Step 2; and,

(b) A set of programs that interpret complex stylus motions as definitions of movement and rhythm, such as the one in Step 5. The programs constitute a subsystem for defining and refining global dynamic descriptions, which may be written in APPL.

(3) The construction of special-purpose animation subsystems, such as the picture-driven animation subsystem in Step 3; the building of special-purpose subsystems, such as the one in Step 6, which aid particular picture definition tasks by facilitating the execution of repetitive actions and the calling of existing routines with suitable parameters.

We have also seen, in Step 8, that APPL allows the activation, for concurrent (quasi-parallel) execution, of sets of programs that compute new animation subsequences and display (play back) existent ones.

II.A.6. WHAT IS TO COME, AND WHAT IS NOT TO COME, AND WHY

This chapter has motivated and established guidelines for the design of APPL, an open-ended, multi-purpose, interactive animation programming system. The next four chapters present an informal outline of the proposed system and the programming language embodied in it. Our major purpose is to suggest some linguistic constructs with which one can directly manipulate structured pictures at an interactive console, and with which one can express algorithms that dynamically transform these images. We have attempted to develop constructs that appear most natural and powerful in aiding interactive computer-mediated animation, and in realizing the guidelines established in II.A.3.

In order to write down a description of APPL, we have had to adopt a precise form in which statements could be expressed, and a particular set of commands to constitute a base language. These are not to be regarded as fixed, for further modifications of such design details will accompany an implementation and empirical evaluation of APPL, activities which are beyond the scope of the dissertation. The dissertation also does not include an extended-BNF, canonic system, or similar formal presentation of the syntax and semantics of the language, since this would be premature and of little enduring value.

Aspects of the design are closely related to several areas of current research in programming languages--techniques for language extension, simulation languages, and complex data structures. There are many difficult problems in these areas that have not yet been solved and that are not solved in this dissertation. We supplement pointers to the relevant literature with an attempt to demonstrate, through intuitive arguments and simple programming examples, the relevance of these issues to computer animation and to the design of APPL. We indicate where important areas of detail have been left incomplete, for example, whether or not there should be run-time block structure, which activity scheduling mechanisms are most useful for animation, and how the chosen data structure can be efficiently implemented. The next researcher will have to tackle these questions; we seek only to justify that these are the ones he should ask.

FOOTNOTES -- II.A.

(*1) BEFLIX has been used primarily in the production of computer-animated technical and expository films, such as References 6, 15, and 20.

(*2) In the case of CAFE, and most likely also in the case of BEFLIX, economic constraints ruled out a system having direct graphical interaction. Nolan and Yarbrough explained their goals as follows:

> There are a number of limitations to this experimental system, perhaps the greatest of which is the lack of graphical interaction with the construction process. This limitation is imposed by a design requirement that the expensive direct-view CRT not be a part of the minimum hardware needed to run the system.[26]

(*3) APPL is more general than JOSS,[65] however, in that the latter does not allow defined commands to be used in higher-level commands. A new conversational language, with excellent, carefully documented design goals akin to those of APPL, is described in Reference 66. Weizenbaum[67] and Licklider[68] have supplied articulate introductions to the potentialities inherent in the use of conversational languages.

(*4) This goal is essentially that expressed by Roberts in Reference 69.

(*5) Thus we transform what Licklider[68] calls a "procedure-oriented language" into a number of different but related "problem-oriented languages." This is how APPL meets the need noted by Knowlton and Huggins[70] for "either an overall complete language for computer animation or a consistent set of intercommunicating modules, with well-defined and rigorously described interfaces between them." With Cheatham, et al,[71] we feel that all aspects of such a language cannot be planned in advance, that the so-called "shell approach" is a priori doomed to failure. As in ELF[71] we adopt the "core approach," allowing the definition of a family of languages through appropriate extensions of the core, or base language, of APPL. The first part of Reference 71, and Reference 72, are readable discussions of the philosophy of language extension.

(*6) The use of an example involving regular abstract figures does not imply that APPL is a priori better suited for this application than for one involving free-hand sketching. The examples throughout Part II underscore this point.

(*7) It is immaterial whether the command is specified by typing its name or an abbreviation for it, by writing the abbreviation to a character recognizer,[58] or by selecting one of a menu of light buttons located somewhere on the scope. We recall the problem of the geometrical layout of these stylus targets on the display surface, discussed in I.A.3. and particular in Footnote 5--I.A. The problem is more severe in APPL than in GENESYS, because the open-ended character of the system requires that the light buttons be augmented whenever a new command is defined. Paging may be the only viable solution.

(*8) We have seen one application of such a program in the syntheses of the pulsating heart, I.D., Approach 4, Technique 2.

(*9) Here the terms 'local' and 'global' refer to the scope of a program variable, or the range of program statements where the variable's name is bound to a particular value, as is explained in II.B.3, rather than to the relative number of frames of a dynamic sequence that are affected by a pictorial operation, as in 'global dynamic description.'

II.B. THE INCORPORATION OF A [illegible]

(*6) The use of an example involving regular abstract figures does not imply that APPL is a priori better suited for this application than for one involving free-hand sketching. The examples throughout Part II underscore this point.

(*7) It is immaterial whether the command is specified by typing its name or an abbreviation for it, by writing the abbreviation to a character recognizer, or by selecting one of a menu of light buttons located somewhere [illegible] geometrical [illegible] display surface, [illegible] in Footnote 5--I.A. The problem is more severe in APPL than in GENESYS, because the open-ended character of the system requires that the light buttons be augmented whenever a new command is defined. Paging may be the only viable solution.

(*8) We have seen one application of such a program in the syntheses of the pulsating heart, I.D., Approach 4, Technique 2.

(*9) Here the terms 'local' and 'global' refer to the scope of a program variable, or the range of program statements where the variable's name is bound to a particular value, as is explained in II.B.3, rather than to the relative number of frames of a dynamic sequence that are affected by a pictorial operation, as in 'global dynamic description.'

II.B.1. COMMANDS AND PICTORIAL DATA IN APPL

APPL, like SKETCHPAD, GENESYS, and other interactive computer graphics systems, permits the user to add and delete points, lines, and other picture items to and from a pictorial data base. What is visible on the scope is changed by commanding the computer to alter the representation of the picture in the data base. A data base is constructed in a working session with APPL, and may be saved and restored from one session to the next.

More specifically, a pictorial data base consists of a structured collection of data items (DATA) and a collection of relations (ATTRIBUTES) which establish some data as PROPERTIES of others. The data base contains what is conventionally called the inner or fundamental or structural representation of a picture, consisting of all the information about the picture that is stored in the computer. There is usually another distinct pictorial data base in a computer graphics system, one we shall not consider here, containing the outer or visual representation of a picture. This consists of only that information that is required by the hardware for the immediate display of the picture.

Data in APPL are of two kinds, pictures and numbers. Pictures may be either pictorial primitives or pictorial aggregates; numbers are represented by scalars and scalar aggregates.

A PICTORIAL PRIMITIVE is a fundamental picture building block, such as a POINT, which is included in the APPL base language in the sense that there are commands for its creation and deletion and for the assignment and modification of its properties. A PICTORIAL AGGREGATE is a structured collection of pictorial primitives. To simplify the following presentation, the only picture "type" represented by a pictorial primitive in the APPL base language will be the POINT; all other picture types, such as a LINE, will be represented by pictorial aggregates as defined through extensions of APPL.

A SCALAR is a single number. We shall for simplicity consider all scalars as real numbers, since the customary computer distinction among integer, fixed-point, and floating-point numbers is well understood and its inclusion would add nothing to this discussion. A SCALAR AGGREGATE is a structured collection of scalars. (*1)

APPL, therefore, is a medium through which the animator creates and manipulates data items--pictures and numbers. The ability to manipulate a datum requires the ability to reference or retrieve it, and this raises the issue of naming data items. Names are created by associating text-strings (sequences of characters) with existent or newly created data. For instance, APPL contains commands which generate new pictures and aggregates. *NEWPOINT* (abbreviated *NEWPT*), *NEWPICTURE* (*NEWPIX*), and *NEWAGGREGATE* (*NEWAGG*) each may take a text-string as an argument. (*2) If the identifier

(text-string) is already bound to an existing structure, that is, names that structure, then the binding is detached without destroying that structure; in any case, a fresh point, null picture, or null aggregate with the identifier as its name is created. A picture or an aggregate may have several explicit names, as well as being implicitly identifiable by virtue of its position in some aggregate. A datum ceases to exist if all its explicit and implicit references are destroyed. (*3)

To increase the legibility of commands and programs, and to remove the need for separate type declaration of names, we adopt the following rules: The name of a scalar consists of a string of small letters, possibly including numbers, such as 'x', 'yloc', 'rr', 'al', and 'g0'. The name of a scalar aggregate consists of a sequence of capital letters, possibly including numbers, such as 'XX', 'YPATH', 'SELECTIONS', and 'RHYTHM0'. The name of a picture consists of two sequences of characters separated by a period ('.'), such as 'PIX.0', 'P.PCURVE', 'P.XWAVEFORMS', 'PT.0', and 'PT.CENTER'. As will be explained more fully in II.E., the prefix of a picture name, that is the character string before the period, often denotes the "type" of the picture. Specifically, the prefix for a point is 'PT' and for an arbitrary picture is 'P', 'PIC', or 'PIX'.

Typical commands names are " *WAIT*;", "*NEWPT ... at ...*;",

"*SET ... to ...;*", "*PUT ... into ...;*", "*SHIFT ... by ...;*", and "*FORMTRIANGLE among ... and ... and ...;*". Command names are assumed to be uniquely identifiable from the first capitalized word. The words in small letters are optional dummy separators. They may be spread throughout a statement between its parameters, for the purpose of aiding its comprehensibility. Each omitted parameter is indicated above by "...". Since command names are considered "reserved words" by the system, they are distinguished in their written form by the use of italics.

A COMMAND is the designation of a command name followed by its required number of parameters, which can be zero or arbitrarily many. A variety of techniques can be used to specify parameters in an interactive environment. For instance, parameters that are visible pictures may be designated by pointing at them with the stylus; if invisible, their names may be typed into the system. Examples of typical commands are "*WAIT;*", "*NEWPT* PT.A *at* 0.0,0.2;", "*SET* x *to* 0.5;", "*PUT* P.A *into* P.B;", "*SHIFT* P.C *by* 0.5,0.5;", and "*FORMTRIANGLE among* PT.A *and* PT.B *and* PT.C;".

II.B.2. EXTENDING THE COMMAND STRUCTURE

Pictures are constructed by directing the system to execute some of its library of commands. The example in Step 1, II.A.4., illustrates the production of a triangle by the direct designation of a sequence of six commands. Since the construction of a triangle is to occur again and again, APPL should have single commands to facilitate this operation. If these do not currently exist, the user can increase the system power by defining them.

Three such extensions of the command structure are listed in II.A.4. The user designates an ordered, numbered, sequence of commands and requests that they be saved as a unit for execution at a later time. These units are called PROGRAMS, and more specifically PROCEDURES. Commands which may be designated for deferred execution include those which may be immediately executed. Hence the system must be instructed when to leave execution mode and when to enter definition mode, and when later to begin direct execution again. This is done by giving the system meta-command *DEFINE*, followed by the program name and its definition, followed eventually by the system meta-command *END*. (*4)

Much of a practical animation system may through such definitional extension be written in the base language. Commands thus defined are readily available for modifications to suit the demands of individual users. We reiterate the

claim, which the sample programs should substantiate, that the base language is low-level enough that one may flexibly control his style of interaction with the system, yet it can smoothly grow to be high-level enough that one may easily write and draw interesting animated displays. In the examples of II.A.4., the basic triangle construction algorithm expressed by Program 1 is adapted in Program 2 to allow triangles to be built on arbitrary existent points as vertices, and in Program 3 to allow new points to be generated and positioned as well.

The three commands that constitute Program 2, *FORMTRIANGLE*, are not expressed in terms of three specific real points in the pictorial data base. Instead they are expressed in terms of dummy parameters, names that are bound to real pictures, and only then have meaning or value, when the program is called, or activated. Hence the constituent commands will operate upon whichever three points are designated as the program's real parameters. If "*FORMTRIANGLE among* PT.U *and* PT.V *and* PT.W;" is activated, then within the body of the program, that is while the constituent commands are being executed, PT.1 stands for PT.U, PT.2 for PT.V, and PT.3 for PT.W.

All program statements are given unique identifying numbers, or labels. These are expressed as fractions between 0 and 1, ordered according to increasing numerical value. The labels, sans decimal point, may equivalently be viewed

as digit sequences, ordered lexicographically from left to right. Hence, the following sequence is in correct order: .03, .12, .121, .122, .13, .130, .131, .21, .45, .99, .999, .9999.

Since the system is open-ended, the command *FORMTRIANGLE* can be designated for immediate execution, and can also be used in defining higher-level commands. One example of this is Program 3, *DRAWTRIANGLE*. That commands can be nested to arbitrary depth in programs defining new commands implies that in APPL the command language and the programming language are harmoniously integrated. It also implies that we must consider the issue of the "scope" of the names of data items, which is therefore our next topic.

II.B.3. THE SCOPE ISSUE

The scope of a program name is the set of program statements in which it has the same meaning, that is, points to the same scalar, picture, or scalar aggregate. A name may in general have several scopes within a set of subprograms. A name whose scope is the "entire set" of subprograms is called a global variable, and one with several scopes is called a local variable, for its meaning is local to a particular subset of subprograms. (*5)

At least four possibilities exist for dealing with the problem of scope in a programming language. (1) All variables may be made global. (2) There may be global variables and variables local to only the particular subprogram in which they are defined. (3) The scope of local variables may be extended according to the compile-time block structure of the master program. (4) The scope of local variables may be extended according to the run-time, or dynamic block structure, that is, the meaning of local variables of any program calling another is passed on to the called program.

The first possible solution fails because of the frameword of program calls that is developed in the next chapter, since several copies of a particular program may be active concurrently at execution-time. The third possible solution fails because it appears cumbersome to impose a concept of compile-time block structure on a conversational language. It is at this stage of the research not obvious

whether the sizeable implementation difficulties of dynamic run-time block structure are warranted for the purposes of computer animation. Thus the choice between solutions (2) and (4) remains an open question. (*6) In either case, there must exist both global and local pictorial data bases. Individual programs or sets of programs will contain private (local) pictures and aggregates whose existence need only be temporary. Yet when the process of animation is terminated, the global pictorial data base will have been augmented, that is movies will have been made.

Pictures and scalar aggregates are global and exist forever, even from session to session if they possess at least one global reference. Local names may appear in a directly executable command given by the user, including their possible use as a direct program parameters, or within programs. In the former case, since the user functions as the main program, the name has the same meaning in all direct commands. An element with only such a name, however, would disappear at the end of a session. A local name within a program would be reclaimed when the program was terminated. If the named datum were located somewhere in the structure of an aggregate with a global name, the binding of the local name would disappear but the entire structure would remain.

So that the user can implicitly declare which names are global and which are local, a distinction in their naming

conventions is required. We omit making the distinction in this presentation of the language, for it would not be useful here, given the fragmented character of the sample programs.

FOOTNOTES -- II.B.

(*1) The AGGREGATE mechanism, used for structuring both collections of pictures and collections of numbers, is the topic of II.D.

(*2) Other commands which assign names to data items will be described in II.D.2.

(*3) This leads to the implementation problem known as "garbage collection." The system must be responsible for deleting from the data base those items whose explicit and implicit names have all disappeared, for they can no longer be referenced and hence manipulated by the user.

(*4) Also compatible with this framework is what I call "procedure writing after the fact," that is, recalling recent sequences of immediately executed commands and designating them post-facto as procedures. Such a capability would require a mechanism that monitors direct commands, including interactive input from the stylus and other devices, and then transforms its representation of these events into a sequence of written APPL commands. It would further require a mechanism whereby the user could replace the names of certain objects transformed by these commands with dummy procedure parameters. For the results of a current investigation into the problem of monitoring interactive input and converting it to a written form for subsequent reuse including possible modification, see Reference 73.

(*5) We stress again what was noted in Footnote 9--II.A., that this use of the terms 'global' and 'local' differs from their use in expressions such as 'global dynamic description.'

(*6) Adoption of run-time block structure leads to difficulties when programs call themselves recursively, unless explicit stack mechanisms are included in the language. Another approach, which avoids this problem, is that which in many simulation languages allows the explicit linking of one program to the local data base of another program. We refer further to work on simulation languages in II.C.

II.C. THE INTERACTIVE DEFINITION OF DYNAMIC DATA

II.C.1. THE INDIVIDUAL CONSTRUCTION OF EACH FRAME IN THE SEQUENCE

The most straightforward way to apply the methods of static picture construction to the definition of dynamic pictures is through the first approach of I.B., the individual construction of a sequence of frames. The animator invokes a sequence of direct console commands such as:

```
NEWPIX P.1;
ACCEPTSKETCH P.1;
NEWPIX P.2;
ACCEPTSKETCH P.2; .....
```

where *ACCEPTSKETCH* is implemented in the animation language:

PROGRAM #4

```
DEFINE ACCEPTSKETCH P.0;
.11    SETCONTEXT P.0;
.21    STOPIF STYLUSUP;
.22    IF STYLUSOFF then RETURN;
.23    NEWPT PT.0 at STYLUS;
.31    GOTO #.21;
END
```

"*NEWPIX* P.1;" creates a fresh empty picture with name P.1. "*ACCEPTSKETCH* P.1" activates the routine with P.1 passed as a parameter. This call on *ACCEPTSKETCH* allows the animator to sketch a picture of dots (points) free-hand. He can cause the program to terminate when the sketch is completed. The next *NEWPIX* command may then be given.

Statement .11 (from now on, we shall use the abbreviation #.11) is executed as "*SETCONTEXT* P.1.". Its effect is to guarantee that any points or new pictures to be created

are included in Picture P.1. (This statement will be refined in the next chapter.) The function of the loop of #.21-.31 is to track the positioning of the stylus and record it by a sequence of points deposited at stylus locations. We assume the use of a Sylvania tablet[54] with three vertical levels, *STYLUSDOWN* on the surface, *STYLUSUP* in the air, and *STYLUSOFF*, raised high above the surface; alternatively, three states may be simulated by any tablet stylus and a pair of switches. Execution is suspended at #.21, if the stylus is lifted to the UP state. If it is next returned to the surface, looping continues; if it is next lifted to the OFF state, execution of the program terminates (#.22). The program would be more realistic if rather than terminating directly at #.22, it first passed P.1 to another program which would thin and smooth the sequence of points.

A somewhat more sophisticated approach automates the advance from the construction of one frame to the next:

PROGRAM #5

```
DEFINE ACCEPTSKETCHES P.SEQ0;
.11    NEWPIX P.SEQ0;
.21    NEWPIX P.0;
.22    PUT P.0 into P.SEQ0;
.23    ACCEPTSKETCH P.0;
.24    GOTO #.21;
END
```

Activation of this program with parameter P.SEQ results in the formation of a picture P.SEQ (#.11). It is in turn filled with a sequence of pictures (#.22), those successively generated and bound to the name 'P.0' (#.21). Each of these pictures,

which may be interpreted as a frame, is individually sketched as above (#.23). The program is simplified and defective in that no way of exiting from the loop is provided. If this were added, then although an individual frame would no longer be bound to a unique name, it would continue to exist by virtue of its inclusion in the pictorial aggregate P.SEQ.

The reader will recall the claim, made in II.A.4. and II.A.5., that APPL can be used for system-building as well as for directly generating dynamic displays. *ACCEPTSKETCHES*, a simple program written in APPL, when supplemented with a program that plays back P.SEQ, constitutes an elementary, special-purpose, interactive animation system.

II.C.2. THE GENERATION OF FRAMES FROM AN ALGORITHMIC DESCRIPTION OF THE SEQUENCE--THE CONCEPT OF MOVIE TIME

The following program illustrates the algorithmic specification of a changing display, in which a triangle grows through the upward movement of its top vertex (#1):

PROGRAM #6

```
DEFINE GROWINGTRIANGLE;
.11     NEWPT PT.1 at 0.0,-0.9;
.12     NEWPT PT.2 at -0.1,-1.0;
.13     NEWPT PT.3 at +0.1,-1.0;
.21     FORMTRIANGLE among PT.1 and PT.2 and PT.3;
.31     MOVE PT.1 by 0.0,0.1;
.41     GOTO #.31;
END
```

In addition to its lack of a mechanism for termination, this program is a fundamentally deficient definition of an animation sequence. It contains no mention of time, or of the speed of the picture change. The speed at which the triangle grows apparently depends upon the time it takes the processor to execute #.31-.41, a very restricted and machine-dependent interpretation. Intuitively, what we want to do is control the speed of the computation so that each pass around the loop takes, for example, 1/24th of a second. Although many complex calculations cannot be so quickly and reliably completed, the ramifications of this suggestion should be explored.

We define a scale of simulated, or movie time, and control the dynamics of an animated display by organizing picture

change with respect to this scale. In Program 6, we consider it a static program, and call it a PROCEDURE. It is static in the sense that the dynamically varying state of the display can only be regarded as a means to achieve a final static picture, that which results when the program terminates, because the user has no control over the intermediate states.

Suppose that we add to Program 6 the statement

```
.32     PAUSE for 1;
```

The processor executing the augmented program suspends the calculation at #.32, and pauses for 1 unit of movie time to elapse. It then computes until the *PAUSE* statement is next encountered, and waits again. Now there are well-defined intermediate states of the display, specifically, the pictures at successive integer values of movie time. We call these pictures frames. Hence we say that movie time is clocked in frames, that is, one unit of movie time is also called a frame. A program containing references to the passage of movie time, which are used to organize the dynamics of picture change, is considered a dynamic program, and is called a PROCESS.

How long is one unit of movie time? He who activates Program 6 presumably expects that the sequence of frames resulting from the computation will be displayed concurrent to their generation at a rate "reasonable" for viewing, and that the sequence will be available for later reviewing or

"playback." He may designate his own reasonable rate with the command *SETCLOCKRATE* n *msec*. The rate is usually held constant for a calculation, and is often (1000 msec/24 frames) = 41 msec. This rate is the desired relationship between movie time and real computation time. If the program is simple enough, the desired correspondence can be achieved, and the processor waits during each cycle for a full 41 msec to elapse. However the actual relationship often deviates from this in an unpredictable, computation-dependent manner, since movie time cannot advance from unit to unit until the new frame is computed or retrieved, displayed, and its image appropriately recorded for future playback.

What happens if the actual speed at which each image is updated to form a new frame differs erraticly from the desired speed established by *SETCLOCKRATE*? In the case of *GROWINGTRIANGLE*, the animator can still evaluate the correctness of his program, which is a major goal in viewing directly the results of the computation. Consider, however, the following program, obtained from *ACCEPTSKETCH* by the addition of one statement:

PROGRAM #7

```
DEFINE ACCEPTPCURVE P.0;
.11    SETCONTEXT P.0;
.21    STOPIF STYLUSUP;
.22    IF STYLUSOFF then RETURN;
.23    NEWPT PT.0 at STYLUS;
.24    PAUSE for 1;
.31    GOTO #.21;
END
```

This is a process that accepts dynamic sketches of parametric curves, or p-curves. It is obviously useless if arbitrary and unpredictable delays are encountered while generating successive points of the curve. Hence, while computing the p-curve, the system records the real time intervals between successive executions of the *PAUSE* statement. Suppose that these are 60 msec., 90 msec., 50 msec., 75 msec., etc., and that the clock rate has been set at 41 msec. The processor introduces artificial delays so that the computation cycle time will appear to equal the maximum value of the clock time and the cycle times already encountered in calculating the sequence. If, as occurs in this example and often in practice, there is regularity in what is computed from frame to frame, then the cycle time will appear uniform although slow. 100, even 200 msec. may be satisfactory for *ACCEPTPCURVE*, provided the user is certain that the time will be approximately stable from point to point, from frame to frame. These considerations are fundamental for all dynamic ("real time") interactive input.

We have remarked in passing that successive frames are automatically saved for future playback. Of course one can compute and then play back sequences of images without using a process. Consider the procedure *ACCEPTSKETCHES* as an example. It directly defines a sequence of pictures. One can easily write in APPL a routine which plays back these pictures. When a process is used to compute an animation

sequence, on the other hand, the system automatically forms and stores the sequence of images that exist at each frame of movie time. By associating a name with the computation, we associate that name with the resulting sequence of frames, and can request playback of the movie simply by referring to its name (*2) As in BEFLIX, this sequence of frames is also what is eventually recorded on photographic film.

In conclusion, what distinguishes the use of the concept of movie time in interactive computer-mediated animation (APPL) from its use in off-line animation (BEFLIX) is the attempt to control in real time a display of successive frames of the movie, both during computation and playback. Given any hardware and software configuration, however, this attempt will often fail. There will be "compute-bound" movies, where processing and retrieval times are too long for effective real-time viewing during sequence construction. There will be "retrieval-bound" or "display-bound" movies, where retrieval and display times are too long for effective real-time viewing during playback. If, for a particular movie, the real-time display of either computation or playback succeeds, then the interaction will be valuable. If both fail, the only remedy is to factor the algorithms or pictures into smaller pieces. (*3) Factoring algorithms is discussed in the remainder of this chapter, factoring pictures in the following two chapters.

The natural way to factor an algorithm which computes a movie is to define separately the various concurrent dynamic activities of which it is composed. Our next topic, therefore, is the modular or parallel definition of parallel actions.

II.C.3. THE MODULAR DEFINITION OF CONCURRENT DYNAMIC ACTIVITIES

Concurrent dynamic activities are actions which may be conceptualized as occurring "in parallel." Mother knitting in the rocker, Father snoozing in the easy chair, and Brother under the couch twisting Sister's arm are concurrent dynamic activities, for they are distinct picture changes which take place over the same extended interval of movie time. The independent synthesis of parallel dynamic strands may at some stage require coupling and coordination; Brother withholds his affection for Sis's arm until Dad's nodding head attains the requisite resonant frequency. Adoption for use in APPL of the structure of program control known as QUASI-PARALLEL PROCESSING may be supported by analogies to simulation languages and by an analysis of alternative program structures expressing one set of concurrent dynamic activities. The argument we now present suggests that an approach to language design should facilitate the expression of concurrent activities, notes that many of the requisite constructs may be adapted from the domain of simulation languages, and illustrates how quasi-parallel processing is superior to standard sequential processing.

We have seen how the design of GENESYS facilitates the addition of new parallel strands of picture activity. A new cel class and its global dynamic descriptions may be defined without altering existent cel classes or dynamic descriptions.

GENESYS also provides a mechanism that facilitates the communication between and the coordination of these strands, the rhythm description.

Simulation is the verbal or mathematical dynamic modeling of a hypothetical or real system; animation is the pictorial dynamic modeling of a hypothetical or real system. Thus many of the concepts and linguistic constructs developed in the five-ten years of research on programming languages for model-building (simulation languages) can fruitfully be applied to computer animation. (*4) In a statement that reflects the current state of the art in simulation languages, Jones argues,

> Normal collections of interacting programs find the subroutine call mechanism adequate for managing the flow of control between the parts of the program (e.g., subroutines). Even complicated heuristic programs do not require a special calling mechanism, although the flow of control is certainly far from predictable in these programs. What is different about simulation programs? Why is the subroutine calling mechanism not adequate for them?
>
> It is because a simulation model is not just one program, but several programs operation in parallel. Each activity in a simulation model is conceptually executing in parallel with the other activities in the model. . . .
>
> Thus, simulation systems need to have a mechanism for allowing separate programs to be run sequentially but appear to be running in parallel. A mechanism is needed for allowing these programs (e.g., activities) to transfer control among themselves, in a completely unpredictable manner. This is necessary because simulation programs may contain stochastic elements in them which determine when they want to run and for how long. Thus, it is impossible in general to predict when one program may wish to run, and how long

> its execution will take. If both of these factors were fixed it would be possible to specify a sequence of calls from one program to the next program and the standard subroutine calling mechanism would suffice.[74] (Emphasis added) (*5)

The mechanism to which he refers may take several forms; we shall, adopting the terminology of SIMULA,[75] call it QUASI-PARALLEL PROCESSING.

The necessity for such a mechanism may be supported by an analysis of a concrete example from the domain of interactive computer-mediated animation. Consider the process *RECORDRHYTHM*, as expressed by the following program:

PROGRAM #8

```
DEFINE RECORDRHYTHM of buttonnumber as RHYTHMDESCRIPTION;
.11    NEWAGG RHYTHMDESCRIPTION;
.12    WAIT;
.21    WAITUNTIL BUTTON buttonnumber OR TOG buttonnumber;
.22    IF TOG buttonnumber then RETURN;
.23    PUT MOVIETIME into RHYTHMDESCRIPTION;
.24    WAITUNTIL NOT BUTTON buttonnumber;
.25    GOTO #.21;
END
```

RECORDRHYTHM accepts the rhythmical tapping of a push-button and records it as a scalar aggregate representing a rhythm description. The *WAIT* command, defined in the next section with greater precision, signals the system that the routine is a process, holds execution in abeyance until the user signals that he is ready by giving the direct command *GO*, and then turns on the clock of movie time. (*6) In writing the program we have assumed that the console has an array of push-buttons and that the scalar parameter 'buttonnumber', when rounded to the nearest integer value, identifies

one of them. Execution is suspended at #.21 until the selected button is depressed or the corresponding toggle switch is set. In the latter case, execution is terminated (#.22). In the former case, the value of the clock in units of movie time is recorded in the scalar aggregate RHYTHMDESCRIPTION (#.23). The second *WAITUNTIL* command guarantees that one must press and release the button before a new value is recorded. Movie time advances concurrently, independent of the position of the button.

A natural question arises--what happens if one is tapping with several fingers on several buttons, or if several animators are thereby recording their rhythmical interactions? The transfer of control among programs recording each button's behavior is certainly "completely unpredictable." Quasi-parallel processing allows the rhythms to be multiplexed by activating a number of copies of this process to accept input from distinct buttons, that is, by calling it with different values assigned to the dummy parameter 'buttonnumber'. Nothing could be simpler! Alternative approaches, without quasi-parallel processing, require:

(1) The construction of a single routine that processes aggregates of buttons with an elaborate scanning mechanism to identify which are depressed at a given instant;

(2) The construction of a cyclic set of routines, needing modification with the addition of a new button, each

of which would call the next until the last were reached, movie time were updated, and control were passed back to the first routine; or,

(3) The construction of a "main program," needing modification with the addition of a new button, which would keep track of and control the cyclic execution of a set of individual button-interrogating routines.

Each alternative, particularly in the more difficult situations which occur when the parallel activities are not copies of a single process, is decidedly unattractive compared to quasi-parallel processing. Hence we shall proceed to its exposition without further justification.

II.C.4. THE FLOW OF CONTROL AMONG INTERACTIVE APPL PROGRAMS

The first feature of quasi-parallel processing that distinguishes it from conventional sequential processing is the importance of a strict separation between the definition of a program and its use, and the possibility of the concurrent existence of several instances of the same program. Each instance has its own private local data base. A use of a procedure is called an EVENT; a use of a process is called an ACTIVITY. (*7)

Consider the following application of the routine *RECORDRHYTHM*. An animator, programmer, and educator are collaborating on a film. While viewing the current version of one constituent sequence, the picture called P.SCENE, each wishes to record, without interrupting the playback, the frame values of key points that interest him. To achieve this, three copies of *RECORDRHYTHM* are to be run in quasi-parallel execution with a playback routine, one which is implemented in the language as follows:

PROGRAM #9

```
DEFINE VIEWSCENE P.SEQUENCE;
.11    SHOW 'THIS MOVIE IS' P.SEQUENCE at xtitle,ytitle;
.12    WAIT;
.21    FOREACH P.O in P.SEQUENCE DO PART 21;
          .211    SHOW P.O;
          .212    PAUSE for 1;
          .213    HIDE P.O;
END
```

#.11 will produce a display of the title, "THIS MOVIE IS

P.SCENE," at the coordinate values bound to the scalar names 'xtitle' and 'ytitle'. The *FOREACH* statement will step through the pictorial aggregate P.SCENE, executing Part 21, that is, #.211-.213, upon each picture in the aggregate. (Part abc, where a, b, and c are digits, consists of all statements with labels of four or more digits such that the first three digits are a, b, and c. Part 21 may therefore include #.210, #.210000, #.211, and #.2199999999.) The *SHOW* command makes the designated picture visible, the *HIDE* command makes it invisible.

The following five commands must now be given directly to the system:

```
VIEWSCENE P.SCENE;
RECORDRHYTHM of 1 as RD1;
RECORDRHYTHM of 2 as RD2;
RECORDRHYTHM of 3 as RD3;
GO;
```

The first four commands activate processes; the fifth one triggers their quasi-parallel execution. The system knows, from the program's definition, that *VIEWSCENE* is a process. However, its activation must not cause the playback to begin before all the button-monitoring routines are active. A mechanism is required to enable the designation of several activities before the dynamic phase of their combined execution begins. We adopt one such mechanism, patterned after the sequencing set of SIMULA and the agenda of OPS, which we call the AGENDA.

Designation of the first activity triggers the execution of all its statements encountered prior to the *WAIT* command. In this case a single statement, #.11 of *VIEWSCENE*, produces the title on the display scope. The system is then ready to accept a new command. Designation of the next three activities, all instances of *RECORDRHYTHM*, has a similar result--only the *NEWAGG* commands are executed. A good intuitive model is a racetrack where horses can be brought into their starting gates only one at a time, although all must be entered before the race can begin. This phase of the construction of an animation sequence is called the static phase, for complete executions of events constructing static pictures, as well as shifts into define-mode producing new procedures and processes, may be interspersed among initializations of activities. In a static phase the clock of movie time is not running, and the user controls the system through the direct designation of commands.

The *GO* command turns on the clock and triggers the dynamic phase. Processes are now executed as simulated time advances. Under certain conditions, execution is halted and a static phase re-entered. Alterations may then be made to static pictures and programs. The dynamic phase may be resumed by a *CONTINUE* command; alternatively, a new dynamic phase may be initiated by repeating the *GO* command, which turns on the clock after resetting it to zero.

The agenda controls execution during a dynamic phase. Each process activation has placed the corresponding activity at the end of the ordered agenda of current computations. Execution after the *GO* signal proceeds in round-robin fashion as primarily determined by program positions on the agenda. A scan of the agenda uniquely determines the next potential computation, thus avoiding the indeterminacy possible in so-called "parallel" processing. The agenda scan guarantees that all computations relevant to the current instant of movie time are completed before its clock is advanced. The agenda is an intermediary which automates the cyclic bookkeeping to which we alluded at the end of the previous section.

In the static phase of the above example, four activities are placed, in the order of their designation, on the agenda. After receiving the *GO* signal, the system begins to compute the *VIEWSCENE* activity. It displays the first picture of the sequence P.SCENE at #.211, and suspends the routine's execution at #.212. It then considers the next entry on the agenda. #.21 of *RECORDRHYTHM* causes it to check the states of the first push-button and toggle switch. If the switch is ON, the activity terminates and is removed from the agenda. If the button is depressed, the value 1 (the clock time) is recorded in RD1. Execution of this activity is eventually suspended at #.21 or at #.24 and the system considers the next agenda entry. Both it and the third *RECORDRHYTHM* command are similarly processed. The system then compares the

execution time since the *GO* signal was received to the last previously designated "clock rate." If that interval of time has not yet elapsed, the system continues scanning the three button-monitoring programs, looking for an opportunity to continue their execution. If no opportunity arises before the requisite time elapses, the movie clock is advanced by one unit, and the agenda scan begins anew. Execution then resumes at #.213 of *VIEWSCENE*.

The user may at any time push a *HALT* button and begin a new static phase. The contents of the agenda are not affected by the *HALT* command. Clearly a *DEACTIVATE* command, which removes items from the agenda, is also required. The command "*DEACTIVATE ALL*;" clears the agenda. The command "*DEACTIVATE FIRST RECORDRHYTHM*;" removes the first copy of this process. If the command sequence, "*HALT*; *DEACTIVATE FIRST RECORDRHYTHM*; *CONTINUE*;", were given the dynamic phase would continue with only two push-buttons active.

An alternative method of termination consists of including *DEACTIVATE* calls in the processes themselves. This has already been done in one way. The *RETURN* command of #.22 of *RECORDRHYTHM* is equivalent to the command "*DEACTIVATE SELF*;". If #.22 were changed to

.22 *IF TOG* buttonnumber *then DEACTIVATE ALL*;

then the agenda would be cleared whenever any of the three users turned on his toggle switch. Somewhat more useful would be

.22 *IF TOG* buttonnumber *then HALT*;

for it allows immediate resumption, via the *CONTINUE* command, without restructuring the agenda.

A fuller description of quasi-parallel processing and of the role of the agenda requires specifying in detail the conditions that can interrupt the execution of activated programs, the response of the system to each interruption, and the effects that one program may have on the execution of another. Although a full analysis is beyond the scope of the dissertation, we conclude this chapter with a sketch of the relevant issues.

One source of complexity arises because activation of a procedure does not necessarily result in its immediate execution to completion. A number of conditions may cause the interruption of the event's calculation. The value of a program parameter or referenced variable (name) may not be suitably defined. (It may itself be undefined, or it may be bound to another name which is undefined.) A program called within the procedure may not yet be defined. Finally, further execution may require user interaction such as the positioning of the stylus on the tablet surface or the setting of a toggle switch to a particular state or to a new state. Depending upon the nature of the interruption, either all execution is stopped until new input is provided, or only the interrupted event is suspended, placed on the agenda, and its state preserved for future resumption. We shall not here

decide which action is a more appropriate response to each of the variety of interruptions that may occur. The choice is not critical with respect to a static phase, for in either case the user can be immediately notified of the unbound datum name or program name or of the missing input.

In a dynamic phase, however, stopping the execution of only the interrupted program results in significantly different system evolution than does stopping all execution. In the former case, one entry is suspended from the agenda, but cycling and the advance of movie time continue. In the latter case, some user input is required before any further processing can take place. Examples of the former kind are "*PAUSE for* 3;", which suspends the process for 3 units of movie time, and the *WAITUNTIL* commands of *RECORDRHYTHM*. An example of the latter kind, parallelling *WAITUNTIL*, is "*STOPIF TOG* 4;", which stops all execution when the fourth toggle switch is set. Notice that, since events as well as activities may be placed on the agenda, a *STOPIF* command may also appear in a procedure.

Thus the system in a dynamic phase computes on all active programs until it has completed at least one total agenda scan, and until a real time interval greater than the "clock rate" has elapsed. It then increments movie time by one unit, and scans the agenda anew. If any program is stopped, all are stopped until one of the four following conditions occurs. (1) The cause of the interruption may be removed

directly, by designating a target, for example, if a target is awaited; execution then resumes as if the stop had never occurred. (2) If some other input is given, a new agenda scan searches for a previously suspended activity which can now be resumed. If it finds one, and that also resumes the stopped activity, all execution proceeds; otherwise, it stops again. (3) If the *HALT* button is pushed, a static phase is re-entered, from which new commands may be activated and dynamic execution later continued. (4) If the *CONTINUE* button is pushed, execution of the interrupted program is attempted again. (*8)

An activity or event is therefore always found in one of seven states. At the instant of real time at which it is in execution, it is running. If it is on the agenda awaiting execution during the current instant of simulated time, it is active. Instances of programs must run sequentially but are active in parallel. If it has been self-interrupted by a command such as *PAUSE*, or by a *SUSPEND* command given by another activity (see below), it is suspended. If it awaits necessary input such as is the result on encountering a *STOPIF* command, it is stopped. If a new static phase has begun, it is halted. If it has terminated its execution, it is deactivated. (Actually, it can never be found in this state, for it no longer exists.) And finally, as we shall now explain, if it has directly called another

program which is currently in one of the first five states, it is inactive.

The inactive state arises because procedures and processes may themselves activate other procedures and processes. This is done in one of two fundamentally different ways. A direct call causes an immediate transfer of control; the caller is made inactive and its further execution delayed until the called program terminates and returns. Alternatively, an event or an activity may be scheduled rather than directly called. Execution of the scheduler continues; execution of the new program begins later as determined by where it is placed on the agenda. Activities may also change the state of other activities. Calls to *SUSPEND* and *RESUME* an activity have obvious meanings in the light of the above classification of states. Typical commands, illustrating possible mechanisms for referring to events and activities on the agenda, would be "*SUSPEND SELF*;", "*RESUME FIRST on AGENDA*;", "*SUSPEND LAST RECORDRHYTHM*;", and "*DEACTIVATE ALL RECORDRHYTHM*;". (*9) These may be included as statements in a program's code or may be given directly by the user during a static phase.

One can view the various items on the agenda, excluding inactive items, as separate strands of a total computation. Calling a program directly from another has the effect of lengthening one strand; scheduling an event or activity adds

a new strand. An event may be viewed as a strand consisting of a single element; an activity is a strand of multiple elements. Thus there is a clean and intuitive relationship between serial and parallel aspects of a task computation under quasi-parallel processing.

FOOTNOTES -- II.C.

(*1) We assume here, for simplicity, that the lines of the triangle are somehow constrained to move with the vertices.

(*2) In implementation environments such as the TX-2 system, where there are both inner structural and outer display representations of pictures, *ACCEPTSKETCHES* might produce the more detailed representation of the animation sequence; the automatic feature, only a display file.

(*3) We do not assume, incidentally, that all aspects of interactive animation require the full attention of the computer. In a time-shared environment, however, the animator must obtain sufficient attention for his dynamic interactions to be meaningful. Hence the activation of a process, which requires close coordination between the passage of real and movie time, is a signal to the system that the animator must have absolute priority until further notice. The effects of a system implementation on aspects of the real-time computation and playback of animation sequences is further discussed in III.B.

(*4) Conversely, animation could be applied to depict the results of a simulation.

(*5) Jones,[74] pp. 92-93.

(*6) *WAIT* commands, which for simplicity were temporarily omitted from Programs 6 and 7, must appear in all processes. In Program 6 it could be #.14 or #.22, in program 7, #.12.

(*7) Our definitions of these terms reverse those of SIMULA,[75] for we find our choices more intuitive.

(*8) Programmers will realize that the *STOPIF* command may be used to implement a very effective generalized-breakpoint, "break if some condition is met."

(*9) A more complete definition of APPL should include a precise specification of these mechanisms.

II.D. APPL'S DATA STRUCTURE FOR MODELING PICTURES AND GLOBAL STRAMIC [illegible]

(*1) We assume here, for simplicity, that the lines of the triangle are somehow constrained to move with the vertices.

(*2) In implementation environments such as the TX-2 system, where there are both inner structural and outer display representations of pictures, ACCEPTSKETCH[illegible] might produce [illegible] sequence, the [illegible] fills.

(*3) We do not assume, incidentally, that all aspects of interactive animation require the full attention of the computer. In a time-shared environment, however, the animator must obtain sufficient attention for his dynamic interactions to be meaningful. Hence the activation of a process, which requires close coordination between the passage of real and movie time, is a signal to the system that the animator must have absolute priority until further notice. The effects of a system implementation on aspects of the real-time computation and playback of animation sequences is further discussed in III.B.

(*4) Conversely, animation could be applied to depict the results of a simulation.

(*5) Jones,[74] pp. 92-93.

(*6) WAIT commands, which for simplicity were temporarily omitted from Programs 6 and 7, must appear in all processes. In Program 6 it could be #.14 or #.22, in program 7, #.12.

(*7) Our definitions of these terms reverse those of SIMULA,[75] for we find our choices more intuitive.

(*8) Programmers will realize that the STOPIF command may be used to implement a very effective generalized-break-point, "break if some condition is met."

(*9) A more complete definition of APPL should include a precise specification of these mechanisms.

II.D.1. DESIGN CRITERIA FOR A DATA STRUCTURE

The inclusion of a data structure in a picture processing language requires in 1969 no elaborate justification. All data is in fact structured--a "programming bum" in the lowest of low-level languages facilitates his own retrieval of aggregates of data items by mnemonic naming and by lexicographic coding tricks, for example, letting the variable names representing x coordinate values all begin with the letter x. A data structure is simply a mechanism whereby a pre-established coding schemata transfers to the computer some of the burden of appropriately linking related data and making it more easily accessible. A data structure is therefore appropriate for a particular class of problems insofar as it mirrors the patterns of relationships that exist among related data items, and insofar as it facilitates the specification of the most useful accessing paths through the structure. The remainder of this section describes the criteria that a data structure for interactive computer-mediated animation should satisfy. (*1)

(1) The design goals for the data structure should be as independent as possible of implementation considerations. This is particularly important at the current stage of the research, since in designing the language on paper we seek to clarify the requisite concepts and furnish a vehicle for communication about computer animation. Eventually, the

simplicity of the conceptual language described here should be carried through to the operational language, so that the user not be forced to furnish details whose only function is to facilitate translating or running efficiently.

(2) All manipulable substructures should appear to the user dynamically allocable, and capable of enlargement to arbitrary size.

(3) The data structure should facilitate the processing of sequences of data. As we have seen in Part I, sequences of static pictures which form dynamic pictures, spatially or temporally ordered sequences of points or values, and sequences of selections from sequences, all play an important role in interactive computer-mediated animation.

(4) The data structure should facilitate the processing of hierarchies of data. Strong evidence for this requirement is found in the recursive picture processing operations of SKETCHPAD,[39] such as the "move" command which is defined recursively in terms of the moving of subpictures. Only the ability to establish and manipulate complex hierarchic structures will enable complex pictures to be expressed in terms of a few pictorial primitives.

(5) It should be possible to include a single substructure in various (related or unrelated) superstructures. This results in the phenomenon known as sharing, and its function is to provide various paths of accessibility to the single substructure. It is assumed that the system will handle the

"garbage collection" problem, and hence data no longer accessible to the user will eventually disappear automatically.

(6) The mechanisms for accessing or retrieving data from structures should be as flexible and powerful as possible. Accessing may be explicit or implicit. Explicit accessing consists of providing one of the possibly numerous names for a datum. Implicit accessing results from defining a path through a named structure or from defining a search for desired properties among elements of an explicitly named structure.

(7) Finally, the mechanisms for structuring pictorial primitives and numbers should be as similar as possible. Not only will this facilitate learning the language, but it will promote the plasticity in the manipulation of dynamic data to which we referred in Part I, the ability flexibly to transform the representation of dynamic information among numerical sequences, static pictures, and dynamic pictures.

II.D.2. THE AGGREGATE

A single kind of data structure will be used to model both complex pictures and such collections of numerical data as global dynamic descriptions. The data structure is called the aggregate. Aggregates possess characteristics of (ordered) sets--one may test if a datum is a member of a given aggregate, and, if it is a member, obtain its ordinal position. Aggregates possess characteristics of multi-dimensional arrays--one may retrieve an element embedded in a data hierarchy by specifying a finite number of ordinal positions. Aggregates possess characteristics of rings--one may access forward or backwards along a chain of linked data, testing for the beginning or end of the chain, and testing whether a datum is itself an aggregate, that is, a pointer to a lower level ring. The aggregate structure was developed independently, then found to parallel Balzer's excellent yet misleadingly-titled concept of dataless programming.[78]

Balzer describes the goal of dataless programming as follows:

> It conceives of a program as the specification of a set of manipulations to be performed on a set of data values, and that this specification should be independent of the form in which these data values are represented.[78]

By form he means what D'Imperio[76] calls a storage structure, the real physical encoding of a logical and conceptual data structure. The term 'dataless' is misleading in that the

body of a program is to be storage-structure independent, but not data-structure independent. There is a data structure, in APPL called the aggregate, which is a generalization of all ordered hierarchic data representations, including the array, list, and ring. Balzer correctly notes that a programmer who chooses an array early in his coding may at a later stage wish it were instead a list or a ring. The solution is the definition of a data structure so general that it includes the common accessing mechanisms of these ordered hierarchic data representations. However, it appears unreasonable to implement a correspondingly general storage structure. Balzer's solution is:

> Once the user has completed this programming, he can determine for each collection (aggregate) what data representation is suitable for the processing involving that collection.[78]

Thus if the programmer discovers that all references to elements of a particular aggregate are expressed as ordinal positions, he can make the correct implementation choice of an array.

The current definition of the aggregate mechanisms differs from dataless programming primarily in that:

(1) There are no commands with which the APPL user can choose a storage structure to implement a particular aggregate, since issues of implementation have not yet been considered. In view of the first design goal stated in the previous section, we intend to minimize the information of this kind that the user must give the system.

(2) APPL has parallel and non-overlapping aggregation mechanisms for the domain of numbers and for the domain of pictures. I hope that this simplification will help the novice programmer comprehend the structuring of data, since the need for structured collections of numbers and structured collections of pictures should be intuitively reasonable. Mechanisms for associating numbers with pictures are discussed in II.E.

(3) In dataless programming, the user must declare the hierarchic structure of an aggregate and name and declare each sub-aggregate at compile-time. In APPL, any aggregate may be dynamically grown to whatever complexity is desired. (*2) (*3)

The aggregate is best defined through an enumeration of the constructs for its manipulation. These include commands for:

(1) its construction and destruction;

(2) its aggregation and disaggregation;

(3) its reaggregation;

(4) the access or retrieval of a datum from an aggregate;

(5) iterative data access or retrieval;

(6) analysis of aggregate structure.

Finally, the process of the assignment of names to values and the question of structure sharing are discussed.

(1) The construction and destruction of aggregates:

The command "*NEWPICTURE* PIC.A;" (abbreviated "*NEWPIX* PIC.A;") generates a new empty picture named PIC.A. If a picture PIC.A already exists, it is cleared, that is, all its elements are removed. Similarly, "*NEWAGGREGATE* PATH1;" (abbreviated "*NEWAGG* PATH1;") creates a new empty scalar aggregate. "*NEWPIX* PIC.B *as* (PT.A PT.B (PT.C PT.D));" generates a new picture PIC.B, whose first element is PT.A, whose second element is PT.B, and whose third element is the aggregate whose first element is PT.C, and whose second element is PT.D. "*DELETE* PIC.A;" or "*DELETE* PATH1;" totally destroys an aggregate and all its unnamed substructures not included in any other aggregates. All elements of aggregates are actually references, or pointers, to numbers, pictures, and other aggregates. Thus the existence of an element is independent of its relationship to the aggregate, unless it has no explicit name and is included in no other structure. In this case, it should be garbage collected, for it can never be referenced again.

(2) The aggregation and disaggregation of data items:

The command "*INSERT* PT.A *as FIRST of* PIC.C;" inserts PT.A ahead of all existing elements in the aggregate PIC.C. "*INSERT* PT.A *as THIRD of* PIC.C;" leaves the first two elements of PIC.C unchanged, but places PT.A ahead of all other

elements. "*PUT* PT.A *into* PIC.C;" is an abbreviation for "*INSERT* PT.A *as LAST of* PIC.C;". (*4) The command "*TAKE* PT.A *from* PIC.B;" results in PIC.B = (PT.B (PT.C PT.D)), whereas "*TAKE* PT.C *from* PIC.B;" leaves PIC.B unchanged or generates an error. (PIC.B was defined above.)

Since APPL programs will use heavily the capacity to hierarchically structure pictures, we adopt a pictorial context convention to simplify the specification of such commands. Execution of "*SETCONTEXT* PIC.E;" guarantees that each newly generated picture, such as a point created by *NEWPT*, automatically be included as the last element of the context PIC.E. "*REMOVE* PT.A;" is therefore equivalent to "*TAKE* PT.A *from* PIC.E;". The context is kept in stacks, with a unique stack assigned to the set of direct commands and one to each quasi-parallel computation strand. "*SETCONTEXT* PIC.E;" pushes PIC.E onto the top of the appropriate stack. *RESTORECONTEXT* pops the stack, *NOCONTEXT* clears it. (*5) The pictorial context serves as a buffer or working area for the assemblage of new pictorial aggregates. It particularly simplifies the process of aggregation according to the order of creation of new pictures; each is automatically appended to the sequence.

(3) The reaggregation of data items:

Ordering aggregate elements by their instants of creation may later prove unsatisfactory. Sorting routines can easily be written in the language. Because of its wide applicability, however, one automatic sorting capability is provided. If the elements of PIC.F are points, then the command "*ORDER* PIC.F *by* (±)*X*;" rearranges the points of PIC.F according to increasing (±)x coordinate. Any picture attribute defined on the elements of the aggregate can be the ordering criterion. (Picture attributes are discussed in the next chapter.)

(4) The access or retrieval of a datum from an aggregate:

The mechanism for accessing individual elements of an aggregate is called a selector. Examples of selectors are: *FIRST*, *SECOND*, *THIRD*, *6TH*, k*TH*, *LAST*, *THIS*, *NEXT*, *PREVIOUS*, *THIRD AFTERTHIS*, *6TH BEFORETHIS*, and k*TH FROMLAST*. These constructs include the usual accessing mechanisms of arrays and rings. The scalar "k", when evaluated in one of these selector expressions, is rounded to the nearest integer.

Each aggregate, whether explicitly named or not, has an implicit pointer, which implements the 'CURRENT' mechanism of dataless programming. Relative selectors such as *THIS* and *NEXT* are interpreted with respect to this pointer. Upon creation of an aggregate, it points to the first element; it may also be explicitly assigned by a statement such as

"*BEGINAT LAST of* PIC.G;" or "*BEGINAT* PT.A *in* PIC.G;". Still undecided in the language's design is the choice of conventions and commands to aid the user's control of the movement of the implicit pointer. (When scanning the aggregate PIC.G, one would want the reference "...*NEXT of* PIC.G..." to move the pointer of PIC.G to the right, but in other circumstances such automatic movement would be unfortunate.) Explicit pointers, or bugs, may be simulated by the use of scalar variables as selectors, although inclusion of the 'GENERATOR' concept of dataless programming would be more elegant.

Hierarchic accessing paths are achieved by cascading selectors. "...*FIRST of* k*TH of* PIC.G..." retrieves the first element of the kth element of PIC.G. If implementation is not too difficult, varieties of syntactic sugar would be particularly helpful here. Preferable to the bulky "*SECOND FROMLAST of* k*TH AFTERTHIS of* alpha*TH of* PIC.G" would be "PIC.G (alpha, k *AFTERTHIS, SECOND FROMLAST*)". (*6)

(5) Iterative data access or retrieval:

Much of the processing of complex pictures can be expressed in terms of operations iterated over elements of an aggregate. Typical iterative commands are:

```
FOR i=0 STEP 1 UNTIL i>10 DO NEWPT PT.0 at -1.0+(0.2)×(i),0.0;
FOR i=j STEP k DO PART 11;
FOREACH PT.0 in PIC.H FROM PT.A THROUGH THIRDFROMLAST DO #.211;
FOREACH xx in -PATHA UNTIL #(xx∈PATHA)=10 DO -xx→xx;
```

The first statement places a string of 11 equally spaced points along the horizontal axis of the current pictorial context. The second statement, which has no termination condition, loops if PART 11 contains no reference to simulated time; otherwise, it continues executing as long as the clock of movie time is running. The third statement causes the execution of #.211 on successive elements of PIC.H, beginning with that which is also named PT.A, and ending with the third from the last element. The fourth statement negates the values of all but the first ten scalars in PATHA, scanning PATHA from the last element towards the first.

(6) The analysis of aggregate structure:

The ability to interrogate the system about the aggregate aids the accessing of a dynamically changing structure. The primitive function "*SIZE of*...", when applied to an aggregate, yields a scalar whose value is the number of elements of the aggregate. The functions "*IF* PIC.I {*ISNIL/ISATOMIC/ISUNARY /ISMULTIPLE*}..." test, respectively, if PIC.I is empty, is a pictorial primitive, is an aggregate with one element, or is an aggregate with many elements. "*IF* PIC.I∈PIC.J..." tests if PIC.I is an element of PIC.J. If this is true, then "#(PIC.I∈PIC.J)..." yields the ordinal position of PIC.I in PIC.J. "*IF ATSTART* PIC.I..." and "*IF ATEND* PIC.I..." test if PIC.I's implicit pointer is at the first or last element of the aggregate. If PIC.J is an aggregate containing one

element only, then "...↓PIC.J..." yields that element. In any case, "...↑PIC.J..." is a pictorial aggregate whose only element is PIC.J.

(7) The assignment of names to values:

Consider the problem of recording and updating a property of a dynamically changing pictorial aggregate. For example, the use of a picture called a waveform aids the editing of a path description, as we have seen in Chapter I.C. Imagine that the user requires in the pictorial data base a record of which picture point is the absolute maximum of the waveform WAVEA. Available is an attribute *MAXIMUMY*, a function which takes a pictorial aggregate as an argument and returns the desired point as the value. "*MAXIMUMY of* WAVEA..." is to yield PT.MAX. But what does "yielding PT.MAX" mean? What is assigned the name PT.MAX?

One interpretation is that the desired point is to be identified, a copy made, and the copy preserved under the name 'PT.MAX'. This is not suitable, however, if the waveform is later resketched. A second interpretation is that the name 'PT.MAX' is bound to the maximum element of WAVEA. This is acceptable if resketching changes the value but not the ordinal location of the maximum element. It breaks down, however, if the new waveform shape shifts the location of the maximum. A third interpretation, suitable also for this case, is that the name 'PT.MAX' is bound to the picture

expression "*MAXIMUMY of* WAVEA", in the sense that future references to PT.MAX result in a recomputation of "*MAXIMUMY of* WAVEA". (*7)

All three interpretations should be included in the language. Hence "3→x;" results in the creation of a new scalar 3 assigned the name 'x'. "*MAXIMUMY of* WAVEA → PT.MAX;" identifies the desired element of WAVEA, copies it, and assigns to it the name PT.MAX. "*SET* x *to* y;" identifies the scalar named by y, and assigns it an additional name x. "*SET* PT.MAX *to MAXIMUMY of* WAVEA;" identifies the desired element of WAVEA, and assigns it an additional name PT.MAX. If WAVEA were then deleted, its element named PT.MAX would still remain. "*LET* x *be* y+3;" means that future references to x will be evaluated as if "y+3" had been written. "*LET* PT.MAX *be MAXIMUMY of* WAVEA;" means that future references to PT.MAX will be evaluated as if the picture expression had been written.

Can one APPL expression be bound to another expression? We take "*MAXIMUMY of* WAVEA → *FIRST of* P.PTSMAX;" to mean that the desired element of WAVEA is copied, and a reference to the copy is made the first element of the pictorial aggregate P.PTSMAX. We take "*SET FIRST of* P.PTSMAX *to MAXIMUMY of* WAVEA;" to mean that the reference, or pointer, to the element is copied and made the first element of P.PTSMAX. Because elements are references, such sharing of structures can be

allowed. Substituting LET for SET, however, is not allowed, for the introduction of this major complexity appears unjustified.

All three interpretations should be included in the language. Hence "3+x;" results in the creation of a new scalar 3 assigned the name 'x'. "MAXIMUMY of WAVEA + PT.MAX;" identifies the desired element of WAVEA, copies it, and assigns to it the name PT.MAX. "SET x to y;" identifies the scalar named by y, and assigns it an additional name x. "SET PT.MAX to MAXIMUMY of WAVEA;" identifies the desired element of WAVEA, and assigns it an additional name PT.MAX. If WAVEA were then deleted, its element named PT.MAX would still remain. "LET x be y+3;" means that future references to x will be evaluated as if "y+3" had been written. "LET PT.MAX be MAXIMUMY of WAVEA;" means that future references to PT.MAX will be evaluated as if the picture expression had been written.

Can one APPL expression be bound to another expression? We take "MAXIMUMY of WAVEA + FIRST of P.PTSMAX;" to mean that the desired element of WAVEA is copied, and a reference to the copy is made the first element of the pictorial aggregate P.PTSMAX. We take "SET FIRST of P.PTSMAX to MAXIMUMY of WAVEA;" to mean that the reference, or pointer, to the element is copied and made the first element of P.PTSMAX. Because elements are references, such sharing of structures can be

II.D.3. AN ILLUSTRATION OF MODELING COMPLEX PICTURES BY PICTORIAL AGGREGATES

This section presents a concrete illustration of the concept of modeling complex pictures by pictorial aggregates. The example is a process which implements part of a system for "editing," that is interspersing fragments from several distinct pictorial sequences, or 'scenes'. Each scene consists of a sequence of pictures, or frames. Suppose that these sequences themselves are included in a pictorial aggregate called S.SCENES. Suppose further that there is a pictorial aggregate called S.TAPES, each member of which is a picture consisting of a number of points equal to the number of frames in a corresponding sequence of S.SCENES. S.TAPES is a picture that portrays the quantity and length in frames of available scenes. Thus, if there are three scenes in S.SCENES, whose lengths are 20, 25, and 30 frames then there are three 'tapes' in S.TAPES, arranged in rows of 20, 25, and 30 points, respectively.

The editing system is implemented by:

PROGRAM #10

```
DEFINE EDITINGSYSTEM;
.11     SHOW S.TAPES;
.12     SET PT.0 to INPUT;
.13     SET TAPE.0 to ↑ PT.0;
.14     SET SCENE.0 to #(TAPE.0εS.TAPES)TH of S.SCENES;
.15     SET FRAME.0 to #(PT.0εTAPE.0)TH of SCENE.0;
.16     HIDE S.TAPES;
.17     SHOW FRAME.0;
.18     BEGINAT FRAME.0 in SCENE.0;
.19     WAITUNTIL BUTTON 4 OR BUTTON 5 OR BUTTON 6;
.20     HIDE FRAME.0;
.21     IF BUTTON 5 THEN DO PART 21;
            .211    SET FRAME.0 to NEXT of SCENE.0;
            .212    GOTO #.17;
.22     IF BUTTON 6 THEN DO PART 22;
            .221    SET FRAME.0 to PREVIOUS of SCENE.0;
            .222    GOTO #.17;
.23     IF TOG 4 THEN GOTO #.31;
.24     IF TOG 5 THEN RETURN;
.25     GOTO #.11;

.31     PLAYBACK S.EDITEDSEQUENCE;

.41     FOREACH PIC.0 in SCENE.0 FROM FRAME.0 UNTIL BUTTON 7
        DO PART 41;
            .411    SET PTR.0 to PIC.0;
            .412    PUT PTR.0 into S.EDITEDSEQUENCE;
            .413    SHOW PTR.0;
            .414    PAUSE for 1;
            .415    HIDE PTR.0;
.42     GOTO #.11;
END
```

The purpose of calling the system is to modify the edited sequence, the pictorial aggregate S.EDITEDSEQUENCE, which exists in some form prior to the call. The animator views the picture S.TAPES (#.11). To supply the input expected at #.12, he designates an element of one of the tapes with the stylus. (*8) The unique tape containing the designated point is identified at #.13. The scene corresponding to the tape is identified at #.14. #.15 isolates

the frame whose position in the scene corresponds to the location of the point in the tape. This frame is displayed in place of the picture of the tapes (#.16-.17). The implicit pointer of the aggregate representing the current scene is set to point to the current frame (#.18).

Now the animator is expected to respond again. He may step forwards along the scene, viewing each frame (#.21), or he may search backwards (#.22). He may recall the picture of the tapes and designate a new point on any one of them (#.25). He may terminate the execution of the system (#.24). Finally, when he finds a suitable section, he can add it to S.EDITEDSEQUENCE (#.23).

First, S.EDITEDSEQUENCE as now constituted is played back (#.31). Then pointers to successive elements of the current scene, beginning at the current frame (#.41), are inserted at the end of S.EDITEDSEQUENCE (#.412) and made visible for a brief interval of time (#.413-.415). This action is terminated by a button push (#.41), after which the picture of the tapes is again made visible (#.42).

EDITINGSYSTEM as here defined is obviously not a complete editing system; at a minimum there must and can be added facilities for deleting frames from S.EDITEDSEQUENCE.

II.D.4. ILLUSTRATIONS OF MODELING GLOBAL DYNAMIC DESCRIPTIONS BY SCALAR AGGREGATES

Global dynamic descriptions may be modeled, or represented, directly by scalar aggregates. Thus the following program defines the sawtooth path description of II.A.4., Step 4:

PROGRAM #11

```
DEFINE FORMSAWTOOTH PATH of height and period and length;
.11     NEWAGG PATH;
.21     FOR i=1 STEP 1 UNTIL (i>period) or (SIZE of PATH
        = length) DO PUT (height × i/period) into PATH;
.22     IF SIZE of PATH = length then RETURN;
.23     GOTO #.21
END
```

The parameters of *FORMSAWTOOTH* are the name of the scalar aggregate representing the path description, the height of the teeth, the period of the teeth, and the total length of the path description to be computed.

The process *RECORDRHYTHM*, defined by Program 8, illustrates the real-time formation of a dynamic description. Those instants of movie time at which the button is depressed are recorded as successive values of a scalar aggregate representing a rhythm description.

Occasionally we have a pictorial representation of a path description, but require the description itself. For instance, execution of the process *ACCEPTPCURVE*, defined by Program 7, results in the formation of a parametric curve representing a pair of path descriptions. To extract the vertical coordinate variations from this curve, we need

the following:

```
DEFINE EXTRACTY YPATH from P.PICTUREOFPATH;
.11     NEWAGG YPATH;
.12     FOREACH PT.0 in P.PICTUREOFPATH DO PUT Y of PT.0
        into YPATH;
END
```

PROGRAM #12

Y coordinates of successive points in P.PICTUREOFPATH become successive values of the resulting path description. The attribute "*Y of*..." operates on a point and yields its y coordinate. The path description, once extracted from the picture, is represented by the scalar aggregate YPATH. The command *EXTRACTY* would also extract the a path description from its standard waveform representation.

The final example is a procedure which transforms a time-independent selection description and a rhythm description consisting of a sequence of intervals into a time-based selection description. (Recall the definitions in I.C.6.):

```
DEFINE SPREAD SD1 and RD into SD2;
.11     NEWAGG SD2;
.12     BEGINAT FIRST of RD;
.13     FOREACH s in SD1 DO PART 13;
           .131     FOR i=1 STEP 1 UNTIL i > THIS of RD
                    DO PUT s into SD2;
           .132     BEGINAT NEXT of RD;
END
```

PROGRAM #13

II.D.5. PICTURE-DRIVEN ANIMATION IN APPL

In picture-driven animation, we recall, picture change is determined by simple algorithms operating upon collections of static images (cels) and collections of global dynamic descriptions. Commands in the language that cause picture change are called PICTORIAL OPERATIONS. Pictorial operations may be roughly distinguished as determining continuous alterations of value, which we call continuous pictorial operations, and as determining discrete alterations of picture structure, which we call discrete pictorial operations. Individual picture changes, such as occur in the construction of static images, can be produced by pictorial operations with parameters that are scalars. Ongoing picture change, resulting in animated displays, can be produced by continuous pictorial operations whose parameters are successive elements of path descriptions (continuous movement descriptions) and by discrete pictorial operations whose parameters are successive elements of selection descriptions (discrete movement descriptions).

Examples of continuous pictorial operations applied to points are: "*REPLACE X of* PT.A *by* X *of* PT.B; *ALTER* θ *of* PT.C *by* deltatheta; *SCALEUP R of* PT.D *by* expansionfactor;". The first command shifts the point PT.A horizontally and assigns it a new x coordinate identical to that of PT.B. The second command rotates PT.C around the origin by an angle

deltatheta. The third command leaves the angle invariant, but multiplies the distance of PT.D from the origin by the scalar expansionfactor. The meaning of continuous operations when applied to pictorial aggregates will be discussed in the next chapter.

Discrete pictorial operations include assignment statements, *HIDE* and *SHOW* commands, picture generators such as *NEWPT*, and commands that alter aggregate structure, such as *INSERT* and *REMOVE*. Their meaning when applied to pictorial aggregates will also be discussed in the next chapter.

Picture-driven animation may be implemented in APPL by passing cel classes, expressed as pictorial aggregates, and global dynamic descriptions, expressed as scalar aggregates, as parameters to pictorial operations. Recall, from I.A.3., the three critical commands given by the GENESYS animator in constructing the bouncing wedge of 'SPROINGBOINGZAP':

```
FORMCEL 1 in the class P.WEDGE;
SKETCHPCURVE P.WEDGE;
PLAYBACK;
```

These commands are defined by the following extensions of APPL, assuming, to simplify the programs, that picture change is limited to the switching of cels as determined by selection descriptions, and to vertical translational motion as determined by path descriptions:

```
DEFINE FORMCEL n in the class  P.CLASS;
.11    IF P.CLASS εP.CELCLASSES then GOTO  #.21;
.12    NEWPIX P.CLASS;
.13    PUT P.CLASS into P.CELCLASSES;
.21    NEWPIX P.CEL;
.22    INSERT P.CEL as nTH of P.CLASS;
.23    ACCEPTSKETCH P.CEL;
END
```

PROGRAM #14

All cel classes are to be included in the pictorial aggregate P.CELCLASSES. Unless the class already exists (#.11), a new empty class is formed and put into the aggregate (#.12). Then a new cel is formed, appropriately inserted into the class (#.22), and finally sketched by the animator (#.23). (*9)

```
DEFINE SKETCHPCURVE P.CLASS;
.11    NEWPIX P.PCURVE;
.12    ACCEPTPCURVE P.PCURVE;
.21    EXTRACTY YPATH from P.PCURVE;
.22    #(P.CLASSεP.CELCLASSES) → n;
.23    INSERT YPATH as nTH of YPATHS;
END
```

PROGRAM #15

A parametric curve is initialized (#.11), then sketched in real time by the animator (#.12). (*10) The variations of the vertical coordinate with time are extracted to form a path description (#.21). (*11) The path description is inserted into a scalar aggregate YPATHS, whose elements are arranged to correspond to the cel classes in P.CELCLASSES (#.22-.23).

And, finally:

PROGRAM #16

```
DEFINE PLAYBACK;
.11     WAIT;
.21     FOREACH P.CLASS in P.CELCLASSES DO PART 21;
            .211    #(P.CLASSεP.CELCLASSES) → n;
            .212    MOVIETIMETH of nTH of SDESCRIPTIONS → s;
            .213    MOVIETIMETH of nTH of YPATHS → ylocation;
            .214    SET P.VISIBLECEL to sTH of P.CLASS;
            .215    REPLACE Y of P.VISIBLECEL by ylocation;
            .216    SHOW P.VISIBLECEL;
.22     PAUSE for 1;
.23     FOREACH P.CLASS in P.CELCLASSES DO PART 23;
            .231    #(P.CLASSεP.CELCLASSES) → n;
            .232    (MOVIETIME-1)TH of nTH of SDESCRIPTIONS → s;
            .233    HIDE sTH of P.CLASS;
.24     GOTO #.21;
END
```

Corresponding to each class is also a selection description contained in the scalar aggregate SDESCRIPTIONS. At each instant of movie time, elements of the selection description (#.212) and of the path description (#.213) associated with each cel class are retrieved. The element of each selection description chooses a cel from the corresponding class (#.214). Before being displayed (#.216), its center is translated by the element of the corresponding path description (#.215). At the next instant of movie time, the currently visible cels are first removed (#.23), before the image for that frame is constructed (#.24).

FOOTNOTES -- II.D.

(*1) References 76 and 77 survey a variety of data structures and languages for their manipulation.

(*2) Problems of implementation will be made severe by this generalization.

(*3) We also do not introduce, although we may later incorporate, something comparable to Balzer's nice canonical form for data and function references, whose purpose is to minimize the syntactic differences between expressions that retrieve and compute information:

> He (the user) may also decide that a piece of information should not be supplied by a data reference on a collection (aggregate), but should be supplied by a function through a calculation or series of calculations on other information. ...[78]

(*4) What should happen if the datum already belongs to the aggregate is an open question. We favor considering it an error. Thus there can be no aggregate (PT.A PT.A), although PIC.D = (PT.A PT.B (PT.A PT.C)) is legal.

(*5) To preserve the simplicity of the core language, the CONTEXT mechanisms should not be included in it, but rather be implemented via definitional extension of APPL.

(*6) See pp.7-8 of Reference 79 for a discussion of the syntactic sugar of accessing paths.

(*7) The three interpretations correspond to the concepts of "call by value," "call by reference," and "call by name," respectively.

(*8) In principle, he could also type a name for the point, or write an abbreviation of its name to a character recognizer.

(*9) Recall the definition of *ACCEPTSKETCH* in Program 4.

(*10) Recall the definition of *ACCEPTPCURVE* in Program 7.

(*11) Recall the definition of *EXTRACTY* in Program 12.

picture description capability of his language. (*1)

The Meaning of Attributes and Properties

The major new concepts are the attribute and properties of pictures. An ATTRIBUTE is a function defined on a picture, one that yields a single value which may be a scalar, scalar aggregate, or another picture. (*2) The value of an attribute of a picture is called a PROPERTY. Properties may be retrieved as stored values, or computed from programs defining

II.E.1. ATTRIBUTES AND PROPERTIES OF PICTURES

The last chapter demonstrated that the ability to create and manipulate complex pictures is enhanced by organizing them into structures called aggregates. In this formalism the animator may express some intuitive concepts of picture description--the inclusion of one picture in another and its consequent removal, the ordering of a sequence of pictures and the accessing of such elements as the first, last, and next of the sequence, and the establishment of a hierarchy of embedded pictures. Many intuitive concepts of picture description, however, may not be expressed gracefully with these mechanisms--the "type," or classification, to which a picture belongs, its location, orientation, or size, its visibility or invisibility, and the distance between it and another picture. The purpose of this chapter is to augment the formalism so that the animator may extend and enrich the picture description capability of his language. (*1)

The Meaning of Attributes and Properties

The major new concepts are the attributes and properties of pictures. An ATTRIBUTE is a function defined on a picture, one that yields a single value which may be a scalar, scalar aggregate, or another picture. (*2) The value of an attribute of a picture is called a PROPERTY. Properties may be retrieved as stored values, or computed from programs defining

the meaning of attributes. Some attributes are primitive, that is, built-in, to the base system; others can be defined through language extension.

There are, roughly speaking, three kinds of attributes: CONTINUOUS ATTRIBUTES, which are mappings from pictures to real numbers (scalars), or sequences of real numbers (path descriptions represented by scalar aggregates), DISCRETE ATTRIBUTES, which are mappings from pictures to integers (scalars) or sequences of integers (selection descriptions represented by scalar aggregates), and RELATIONAL ATTRIBUTES, which are mappings, that is, relations or associations from pictures to other pictures. Each kind will be illustrated by a reasonable choice of primitive attributes for a simple base system whose only pictorial primitives are points.

Continuous attributes express, roughly speaking, the precise geometrical and textural determinants and qualities of a picture. Built-in to the base system are attributes of *I* (intensity), *X* and *Y* (rectangular), and *R* and θ (polar) coordinates of points. Thus "*X of* PT.A", if PT.A is a point, yields a scalar whose value is the x coordinate of PT.A. The system also guarantees that the continuous pictorial operations introduced in II.D.5. correctly set the values of these continuous attributes of points. For instance, both "*ALTER I of* PT.A *by I of* PT.A;" and "*SCALEUP I of* PT.A *by* 2;" result in a doubling of the intensity of PT.A. As we shall discuss below, geometrical coordinates of each

point are measured relative to the position of another point; this is controlled by the primitive relational attribute *"ORIGIN of ..."*.

Examples of continuous attributes that may be defined by language extension are the *"SIZE of"* a picture, the *"DISTANCE between"* two pictures, the *"AREA inside"* an enclosing figure, and the *"DENSITYOFPOINTS within"* a region.

The base system contains no continuous attributes defined on pictorial aggregates, for there appears to be no single interpretation acceptable for all applications. Consider, for example, the concept of the x coordinate, or horizontal position, of a picture composed of a number of points. The most reasonable interpretation of its location is the x coordinate of the center of gravity of the figure. Another user, however, might wish to consider the intensities in the calculation, for example, by computing the "center of mass." Still another may prefer, in the case of a triangle, the x coordinate of that vertex whose horizontal position is between that of the other two. Extensions of the meaning of existent continuous attributes, and the introduction of new ones are discussed in Section II.E.3.

Discrete attributes express, roughly speaking, classifications of pictures into differing categories of structure, for example, primitive or aggregate, and "type", for example, point, rectangle, square. Discrete attributes also represent classifications of pictures into conceptually discrete

abstractions of their properties, for example, growing, shrinking, or remaining constant in size, or outside of, inside of, or overlapping a particular closed figure. There are three primitive discrete attributes: *STRUCTURE*, *TYPE*, and *VISIBILITY*. The corresponding primitive properties and their interpretations are given by the following rules:

STRUCTURE of PIC.TURE =	*ATOMIC*	iff PIC.TURE is a pictorial primitive
	NIL	iff PIC.TURE is a pictorial aggregate that is empty
	UNARY	iff PIC.TURE is an aggregate containing only one element
	MULTIPLE	iff PIC.TURE is an aggregate containing more than one element
TYPE of PIC.TURE =	*POINT*	iff PIC.TURE is a point
	AGGREGATE	iff PIC.TURE is an aggregate not also characterized by another defined type
VISIBILITY of PIC.TURE =	*VISIBLE*	iff PIC.TURE is visible
	HIDDEN	iff PIC.TURE is invisible

The following abbreviations may be used:

STRUCTURE of PIC.TURE =	*ATOMIC*	⇔ PIC.TURE *ISATOMIC*
	NIL	⇔ PIC.TURE *ISNIL*
	UNARY	⇔ PIC.TURE *ISUNARY*
	MULTIPLE	⇔ PIC.TURE *ISMULTIPLE*
TYPE of PIC.TURE =	*POINT*	⇔ PIC.TURE *ISPOINT*
	AGGREGATE	⇔ PIC.TURE *ISAGGREGATE*
VISIBILITY of PIC.TURE =	*VISIBLE*	⇔ PIC.TURE *ISVISIBLE*
	HIDDEN	⇔ PIC.TURE *ISHIDDEN*

The interpretation of the discrete attributes *STRUCTURE* and *TYPE* will be developed in great detail in the next section.

Obviously they are defined on pictorial aggregates as well as on primitives. A point is *VISIBLE* when created, *HIDDEN* after being used as the argument of a *HIDE* command, and *VISIBLE* once more after being used as the argument of a *SHOW* command. Because there is no single "best" interpretation of the *VISIBILITY* of a pictorial aggregate, its meaning must be defined by language extension. Extensions of the meaning of existent discrete attributes, and the introduction of new ones are discussed in II.E.3.

Relational attributes express, roughly speaking, associations between a picture and another picture or aggregate of pictures. The only primitive relational attribute is "*ORIGIN of*...". Execution of the command "*SETORIGIN* PT.A;" implies that PT.A will from then on be the origin of all newly created points. The command "*MEASURE* PT.B *relative to* PT.A;" causes the coordinates of PT.B and all points whose coordinates depend upon PT.B to be recomputed relative to PT.A. Execution of "*NOORIGIN*;" determines that, from that time on, all coordinates of new points are measured relative to the system origin, an invisible, unmoveable point named *SYSTEMORIGIN*.

Some useful relational attributes that could be defined by language extension are the "*HYPOTENUSE of*" a triangle, the "*CENTER of*" a circle, and the "*MAXIMUMY of*" a waveform, that is, its absolute maximum.

One major role of attributes is enabling the implicit specification of all those elements in a picture possessing

certain properties. The implementing mechanism is called a picture selection function, and it may be used wherever a picture is expected. Two typical examples, in a hypothetical extended version of APPL, are "...*ALLTHOSE* TRI.0 *in* P.TRIANGLES *suchthat* AREA *of* TRI.0<9...", and "*ALLTHOSE* PIC.0 *in* PIC.TURE *SUCHTHAT* PIC.0 *ISMULTIPLE AND ORIGIN of FIRST of* PIC.0= *SYSTEMORIGIN*...". Application of a picture selection function upon a pictorial aggregate yields a new aggregate consisting of all those elements from the original aggregate which satisfy the given criterion. The elements are ordered in the new aggregate as they were originally. The result may, of course, be nil, unary, or multiple. Execution of the statement "*ALLTHOSE* PIC.0 *in* PIC.TURE *suchthat* PIC.0 *ISTRIANGLE* → P.TRIANGLES;" causes copies of all the triangles in PIC.TURE to be made and included in a new aggregate P.TRIANGLES. (This assumes that *TRIANGLE* has been defined as a new picture *TYPE*, that is, a new value of the attribute *TYPE*.) Upon execution of "*SET* P.TRIANGLES *to ALLTHOSE* PIC.0 *in* PIC.TURE *suchthat* PIC.0 *ISTRIANGLE*;", a new aggregate, containing all of the triangles that exist in PIC.TURE, is formed and called P.TRIANGLES. (Here, as in II.D.2., the references to the triangles are copied, and not the triangles themselves.) "*FOREACH* PIC.0 *in* PIC.TURE *WHILE* PIC.0 *ISTRIANGLE DO PART* 11;" may be used if elements of the implicitly defined aggregate are to be accessed but no new aggregate is to be formed.

This statement causes PART 11 to be executed on each triangle in PIC.TURE.

How Attributes are Evaluated

Properties, as we remarked at the beginning of this section, are sometimes obtained by retrieval of stored values, other times by computation. If the meaning of an attribute operating upon a particular class of picture is built-in to the base system, then the mechanism of obtaining the property is totally under APPL's control. Therefore, associated with each point is a data block containing the *X*, *Y*, *R*, θ, *I*, *ORIGIN*, *STRUCTURE*, *TYPE*, and *VISIBILITY* coordinates of that point. Similarly, associated with each pictorial aggregate is a data block containing the *STRUCTURE* and *TYPE* of that aggregate. The system assumes responsibility for maintaining correct values for each of these properties. For example, if the command "*ALTER R of* PT.A *by* 2.0;" is executed, then the values of *X of* PT.A and *Y of* PT.A are automatically updated as well. (*3)

If the attribute is defined by extension, or if its meaning upon a particular class of picture is defined by extension, then the property is usually obtained by executing the program which implements the language extension. For the sake of efficiency, however, the user should have the ability to request explicitly that a property be stored and until

further notice retrieved rather than recomputed. The aspect of APPL which allows a user to make these requests depends strongly on the implementation and on APPL operating procedures, and hence is not yet well defined. In III.B., however, we shall briefly discuss the problems raised.

How Properties are Assigned

Some properties can be assigned directly by pictorial operations. The meaning of "setting a value of a pictorial attribute" depends upon whether the attribute is continuous, discrete, or relational, whether it is built-in or has been introduced by definitional extension, and whether the picture's type is primitive or has been defined by a language extension. Furnishing APPL with single commands that in each case assign properties is a difficult problem, one which is related to the issue of "constraint satisfaction." (*4) We begin our discussion here, and continue it in the next two sections.

Consider the following four superficially similar statements: "*REPLACE X of* PT.A *by* 0.5; *REPLACE DISTANCEFROMWALL of* PT.A *by* 0.5; *REPLACE X of* TRIANGLE.A *by* 0.5; *REPLACE AREA of* TRIANGLE.A *by* 0.5;". The first statement is meaningful in the base system, for it refers to a primitive attribute of a primitive picture type. In order that the second statement be meaningful, the user must have defined by extension both the attribute *DISTANCEFROMWALL* and the continuous pictorial

operation which sets it to a particular value. The former definition involves subtracting the x coordinate of PT.A from the x coordinate of 'the wall'; the latter definition involves a recalculation of the X, R, and θ coordinates of PT.A. so that it be located the appropriate distance from the wall. Interpretation of the third command requires extension of the meaning of the attribute X to apply to all 'triangles', assuming that *TRIANGLE* has been defined as a new picture *TYPE*. The fourth statement is similar, requiring definition of a totally new attribute.

Prerequisite to understanding how these definitions are made is a better conception of the meaning of the discrete attributes *STRUCTURE* and *TYPE*. That discussion, to which we now turn, also contains an initial example of the extension of meaning of an attribute, that is, the introduction of new values of *TYPE*.

II.E.2. THE DEFINITION OF NEW PICTURE TYPES

We begin by clarifying the distinction between *STRUCTURE* and *TYPE*. The tests for values of *STRUCTURE*, "*IF* PIC.A *ISATOMIC*...", or "...*ISNIL*...", or "...*ISUNARY*...", or "... *ISMULTIPLE*..." can be used to ascertain the current "structural state" of a changing picture. One may question, for example, if the aggregate formed by a picture selection function contains no elements, a unique element, or many elements satisfying the desired criterion. The same tests can be used in exploring a complex hierarchic aggregate, such as one representing a picture of a maze or a network.

The function of the *TYPE* attribute is to distinguish classes of pictures which possess individual characteristic features, and specialized techniques for their construction, decomposition, and manipulation. The concept of picture "type" is one to which we are accustomed and which we find natural--we think of 'triangles' as differing from 'rectangles', and of an 'isoscoles triangle' as a special type of triangle. Searches or other iterative operations may conveniently be limited by indicating the type of picture to which the search should be restricted, e.g., "...*ALLTHOSE* PIC.0 *in* PIC.TURE *suchthat* PIC.0 *ISTRIANGLE*...", or, expressed in abbreviated form, "...*ALLTHOSE* TRIANGLE.0 *in* PIC.TURE...". Attributes assume different meanings when applied to pictures of different type, e.g., the "*AREA of*" a circle is defined by one

computation algorithm, the "*AREA of*" a triangle by another. Pictorial operations, when applied to two pictures of different type, may similarly require two unique interpretations. Consider, for example, an algorithm that shades (fills with some texture) a square, and another that shades an arbitrary closed figure which may include both convex and concave portions.

To make usable a new picture class that is defined in terms of primitive types, the extended language must contain mechanisms for constructing members of the new class from suitable components, mechanisms for selecting distinguished components, that is, decomposing a picture of the new class, and mechanisms for testing if an arbitrary picture belongs to the new class. Landin[83] has called these mechanisms constructors, selectors, and predicates, respectively. Standish,[84] significantly advancing the work of Landin and others, then presented

> ...a method for defining and manipulating several varieties of data structures. This method consists of embedding a descriptive notation for data structures within a programming language in such a way that the resulting language behaves as a synthetic tool for describing and constructing data and programs in a variety of application areas.

In order that

> ...we can organize our transactions with computers to achieve multiplication of effect without multiplication of effort...,

Standish proposed the goal that

> ...This definition facility should permit the programmer to describe with ease the data structures that are critical to a task and to formulate with ease the operations over these data structures that are necessary for describing the processes he intends to carry out. (*5)

Standish's method, presented in a comprehensive and convincing dissertation, seems to achieve this goal, for when the user defines a new data structure [a new picture *TYPE*, in our formalism], the system automatically provides the associated constructors, selectors, and predicates. (*6) Furthermore, it is precisely these constructors, selectors, and predicates which are used in formulating operations over the new structures. Thus the power of the method stems from economy of specification, that is, what the user obtains automatically when he makes a single picture type definition.

To illustrate the flavor of Standish's approach applied to the definition of a new class of pictures and expressed in our notation, we shall discuss one possible definition of a *TYPE* called *LINE*. The definition depends upon a choice of aspects of 'LINEness' that must be present if an aggregate of points is to be called a straight line. Suppose that a straight line must contain a distinguished start point and end point, and that all its constituent points must be collinear. Furthermore, it should be legal to define a straight line by specifying all its constituent points, in which case the system merely checks them for collinearity and records the *TYPE* in the aggregate's data block. It should also be legal to

specify only the start point and the end point, in which case the system will compute, according to some algorithm, an evenly spaced sequence of intermediate points. To achieve these results, the following procedure may be used:

PROGRAM #17

```
DEFINEATTRIBUTE TYPE of PIC.TURE tobe LINE;
.11   PIXSTRUCTURE ((STARTPT:POINT)(ANY×POINT)(ENDPT:POINT))
.21   TRANSFORMEDBY
.22      IF SECOND ISNIL THEN DO PART 22;
.221        SET SECOND to CONNECTION of STARTPT and ENDPT;
.222        RETURN;
.23      IF PIC.TURE ISNOTCOLLINEAR then ERROR;
.24      RETURN;
.31   SUCHTHAT
.32      PIC.TURE ISCOLLINEAR;
END
```

The introductory statement of Program 17 indicates that LINE is to be the 'name' of a new picture type. Its basic structure is expressed by #.11, which is what Standish[84] calls an elementary descriptor and Cheatham, et al,[71] call a mode descriptor. The first and third elements are to be aggregates containing single elements, the start point and end point, respectively. The second element is to contain all other points in the line; *ANY* indicates that there may be arbitrarily many of these. The term '*POINT*' represents any picture whose *TYPE* is *POINT*.

#.21-24 constitute what Standish calls a constructor modifier. This is a description of operations to be carried out upon an aggregate, provided that its format corresponds

to the structural pattern of the elementary descriptor, in order to transform it into a *LINE*. There is first a test (#.22) if only the start point and the end point have been input; if this is the case, the system constructs the desired sequence of evenly spaced points intermediate to the pair by calling a relational attribute "*CONNECTION of*" (#.221). The four occurrences of the selectors *SECOND*, *STARTPT*, and *ENDPT* are assumed to apply to the straight line under construction, that is, "*SECOND*" is an abbreviation for "*SECOND of PIC.TURE*;". (*STARTPT* and *ENDPT* have been defined as selectors by their appearance in #.11.) If more than two points have been input in the appropriate format, then the system checks them for collinearity before designating the *TYPE* of the aggregate to be *LINE* (#.23). *COLLINEAR* is a predicate or picture state (see the next section), which is either true or false in describing a particular picture. Its definition, another kind of language extension, guarantees that "...PIC.TURE *ISCOLLINEAR*...", "...PIC.TURE *ISNOTCOLLINEAR*...", "...PIC.TURE *ARECOLLINEAR*...", and "...PIC.TURE *ARENOTCOLLINEAR*..." have the obvious meanings.

#.31-32 constitute what Standish calls a predicate modifier. This is a description of tests to be carried out upon an aggregate, provided that its format corresponds to that of the elementary descriptor, to determine if it can be considered a *LINE*. Thus, in order for an arbitrary picture

to have the *TYPE LINE*, it must not only satisfy the structural pattern of #.11 but also the condition of #.32.

This program functions as a declaration which results in an immediate augmenting of the system to include a constructor *NEWLINE*, two selectors "*STARTPT of*" and "*ENDPT of*", and a predicate, "*ISLINE*". Their meanings can best be deduced from a series of examples:

(1) "*NEWLINE* LINE.A *from* ((PT.A)(PT.B PT.C)(PT.D));" results in the formation of a pictorial aggregate named LINE.A, whose *TYPE* is *LINE*, if the points PT.A, PT.B, PT.C, and PT.D are collinear; otherwise, it produces an error.

(2) "*NEWLINE* LINE.A *from* (PT.A PT.B PT.C PT.D);" results in a format error.

(3) "*NEWLINE* LINE.A *from* ((PT.A)(PT.D));" results in the formation of a new line with a computed sequence of points intermediate to PT.A and PT.D.

(4) The selector "*STARTPT of*...", when applied to a picture of *TYPE LINE*, yields its start point, that is, the point contained in the aggregate's first element.

(5) The selector "*ENDPT of*..." functions similarly.

(6) Assuming that (1) has been successfully executed, "...((PT.A)(PT.C PT.B)(PT.D)) *ISLINE*..." is true. (Compare the order of the points to that in (1).)

(7) However, "...(PT.A PT.B PT.C PT.D) *ISLINE*..." is false.

In the example, the system uses the pictorial format, or elementary descriptor, and the constructor modifier to form a constructor, the pictorial format to form two selectors, and the pictorial format and the predicate modifier to form a predicate. Thus the formalism simplifies the introduction of new structures through the definition of constructors, selectors, and predicates by

(1) providing a compact format in which all three can be expressed; and,

(2) isolating that part of the specification, the elementary descriptor, which is used in forming all three.

We still have much to learn about mechanisms for defining picture types.

Currently, because of the strange format for lines which was adopted for the purposes of this section, the command "*NEWLINE* LINE.A *from* (PT.A PT.D);" generates a format error. With some measure of the similarity of one structural pattern to another, an intelligent system could perhaps guess what was intended. Alternatively, we can go beyond Standish's development and assume that the system attempts to execute the constructor modifier even if it registers a format error in matching the input data to the elementary descriptor. Then, we add the following "patch" to Program 17:

```
.211    IF FIRST ISATOMIC AND SECOND ISATOMIC AND THIRD ISNIL
        THEN DO PART 211;
.2111      SET FIRST to ↑ FIRST;
.2112      SET THIRD to ↑ SECOND;
.2113      TAKE SECOND from PIC.TURE;
```

#.211-.2113 will "fix up" the input aggregate so that its form is correct. By the time #.22 is reached, the picture will have been transformed to ((PT.A)(PT.D)).

Further generalizations are needed. With obvious syntactic conventions, "*NEWLINE* LINE.A *from* PT.A to PT.B;" would be correctly interpreted. More difficult to achieve, however, is "*NEWLINE* LINE.A *from* 0.0,0.0 to 0.1,0.1;", for the system requires a mechanism for automatically transforming properties into points. Even more difficult is "*NEWLINE* LINE.A *from* PT.A *for* 0.5 *at* +120°;", which should generate a point at 0.5 unit distance from PT.A, in the direction +120°, and then construct a line between them. This latter example illustrates the problem of constraints. We still lack a good formalism with which to specify pictures (e.g., straight lines) "implicitly," or "non-procedurally," and a corresponding mechanism which can operate upon such a specification and a collection of pictorial data to transform it into a line, that is, to cause the pictorial data to satisfy the constraints of LINEness. Stated somewhat differently, the problem is that of finding a meta-language for specifying classes of picture structure by what are commonly called picture grammars, and a processor which,

given a grammar for a class of pictures (e.g., straight lines) and a particular candidate for LINEness, can determine whether the candidate does or does not have the requisite structure. (P7)

II.E.3. AUXILIARY DEFINITIONS OF PICTORIAL ATTRIBUTES AND OPERATIONS

We are now ready to pick up some loose ends. Specifically, we shall introduce methods by which the language can be extended to include:

(1) new commands, such as pictorial operations;

(2) new attributes, continuous, discrete, and relational; and,

(3) new picture states, or predicates.

The name of the command, attribute, or state being defined may "clash with," or be identical to, that of an existent command, attribute, or state. The new program is then said to constitute an auxiliary, or augmenting definition of the existent entity. As auxiliary definition would typically be made to refine the interpretation of the entity so that it pertains to a new picture *TYPE*. A family of augmenting definitions of a single attribute or operation, identified by a unique name, is called a "procedure" in GPL[85] and a "generic procedure" in BASEL (ELF).[71] Our concept of augmenting definitions parallels that found in papers on these two "extensible languages."

Auxiliary Definitions of Pictorial Operations

We have already discussed, in II.B., the definition of new commands, and we further assume that the system contains editing mechanisms with which the user can alter the definition

of an existing procedure or process. Under some circumstances, however, adding an auxiliary definition is simpler than modifying an existent one. Consider, for example, the continuous operation of shifting a picture horizontally. The command "*ALTER X of* PT.A *by* 0.2;" is meaningful in the base language. However, "*ALTER X of* LINE.A *by* 0.2;" is undefined. The most reasonable interpretation would be expressed by the following extension:

PROGRAM #18

```
DEFINE ALTER X of LINE.0 by deltax;
.11     ALTER X of STARTPT of LINE.0 by deltax;
.12     ALTER X of ENDPT of LINE.0 by deltax;
.13     FOREACH PT.0 in SECOND of LINE.0 DO
        ALTER X of PT.0 by deltax;
END
```

If "*ALTER X of* LINE.A *by* 0.2;" is encountered after this definition is made, the system first checks the new program to see if it can be applied. Since the *TYPE* of LINE.A is *LINE*, the system descends through its aggregate structure, retrieving in turn each point, and calling *ALTER* with the point as an argument. Now the parameter types of the new definition and of the call do not match, and the system successfully applied the initial, primitive definition. Thus, "shifting a line" is defined in terms of the shifting of its constituent points.

This program, however, is applicable only to straight lines. If the shifting of any pictorial aggregate can be

defined recursively in terms of the shifting of all its constituent elements, the following program implements the operation of shifting any picture:

```
DEFINE ALTER X of PIC.TURE by deltax;                    PROGRAM #19
.11    FOREACH PIC.0 in PIC.TURE DO ALTER X of PIC.0 by deltax;
END
```

When an atomic picture is passed to this procedure as an argument, or is encountered in descending through a structure, the system cannot execute #.11, and so attempts instead to use the first definition.

One advantage of the formalism is that it allows the user to provide non-standard interpretations of such operations. Suppose, for example, that shifting a straight line horizontally by an amount 'deltax' is to mean that the right-most point be shifted deltax, the left-most point remain glued in place, and the intermediate points follow the moving one as if on a rubber band. Assuming that the "STARTPT of" and the "ENDPT of" the line are the outermost points geometrically, the task may be accomplished by:

```
DEFINE ALTER X of LINE.0 by deltax;                      PROGRAM #20
.11    IF X of STARTPT of LINE.0 = X of ENDPT of LINE.0
       THEN RETURN;
.12    IF X of STARTPT of LINE.0 > X of ENDPT of LINE.0
       THEN ALTER X of STARTPT of LINE.0 ELSE ALTER X of
       ENDPT of LINE.0;
.13    TAKE SECOND of LINE.0 from LINE.0;
.14    NEWLINE LINE.0 from LINE.0;
END
```

The program recomputes the location of the moving point (#.12), drops the existent set of intermediate points (#.13), and calls the constructor of straight lines to provide a new set (#.14).

Examples of pictorial operations that directly set the values of primitive relational and discrete attributes are "*MEASURE* PT.A *relativeto* PT.B;" and "*SHOW* PIC.TURE;". The former routines has at least one side-effect--a recalculation of the *X*, *Y*, *R*, and *θ* values of PT.A. Although the meaning of "*SHOW* PT.A;" is unambiguous, "*SHOW* PIC.TURE;", if PIC.TURE is of arbitrary structure, could have several meanings. Perhaps for one application all constituent points should be made visible, whereas, for another, only a particular subset of distinguished points should be shown. Hence the meaning of "*SHOW*..." must be refined through language extension.

Definitions of Attributes and Properties

The problems and solutions in this domain are much the same as those discussed above. The attribute *X*, when applied to a straight line, could be defined by the following:

PROGRAM #21

```
DEFINEATTRIBUTE X of LINE.0;
.11   RETURN ((X of STARTPT of LINE.0)
      + (X of ENDPT of LINE.0))/2;
END
```

If we adopt the convention that the x coordinate of any aggregate is the x coordinate of the first point encountered in descending recursively through the first element of the aggregate, then the following suffices to define the meaning of "*X of*" any picture:

PROGRAM #22

```
DEFINEATTRIBUTE X of PIC.TURE;
.11   RETURN X of FIRST of PIC.TURE;
END
```

Totally new attributes may also be introduced, such as the following relational attribute:

PROGRAM #23

```
DEFINEATTRIBUTE MIDPOINT of LINE.0;
.11   NEWPT PT.MID at X of LINE.0, Y of LINE.0;
.12   SET PT.BEST to FIRST of SECOND of LINE.0;
.13   FOREACH PT.0 in SECOND of LINE.0 DO
      IF DISTANCE of PT.0 to PT.MID < DISTANCE of PT.BEST
      to PT.MID THEN SET PT.BEST to PT.0;
.14   RETURN PT.BEST;
END
```

This program returns that point included in the straight line aggregate that is closest to the geometrical midpoint of the line. It assumes that "*X of*" and "*Y of*" a straight line yield the coordinate values of the geometrical midpoint, as they would if defined as in Program 21.

Definitions of Picture States

We have seen that some new predicates are formed as by-products of type definitions. They can also be defined

directly, as in the following illustration:

```
DEFINESTATE PIC.1 TOTHERIGHTOF PIC.2;                    PROGRAM #24
   IF X0( PIC.1 ) < X0( PIC.2 ) THEN RETURN TRUE
   ELSE RETURN FALSE;
END
```

FOOTNOTES -- II.E.

(*1) Papers on graphic data-base systems[80,81] for question answering and pattern recognition illustrate a richer picture description capability ("the circle is to the right of and bigger than the triangle, and is inside the rectangle") than is customarily found in synthetic computer graphics systems, that is in systems for constructing pictures (SKETCHPAD, BEFLIX, CAFE, GENESYS). The value of describing pictures by a complex set of relations has been amply demonstrated by the ease with which members of the TX-2 community have written and modified interactive graphics programs in the LEAP Language for the Expression of Associative Procedures.[82] LEAP augments the descriptive capability of a data base of associations with the implicit search and retrieval capability of associative processing ("find all those circles to the right of and bigger than the triangle, and inside the rectangle"). One of our goals is to place these powerful expressive tools in the hands of both the user and the builder of an interactive graphics system. This can succeed because APPL is the language of both the user and the system builder, whereas LEAP is the language of only the system builder.

(*2) Here APPL is less general than LEAP, for the latter language allows multi-valued relations between objects. LEAP also considers one universe of items, which may appear as attributes as well as objects, whereas in APPL the set of attributes is disjoint from the set of data items, consisting of pictures and numbers.

(*3) Whether it updates these properties immediately, or only "when necessary," is implementation-dependent and for the purpose of this discussion immaterial.

(*4) See Footnote 5, I.B.

(*5) Standish,[84] pp. 1,3,11.

(*6) The constructors, selectors, and predicates for pictures of primitive *TYPE*, that is for PICTORIAL PRIMITIVES, are built-in to the base system of APPL. The constructor for a *POINT* is *NEWPT*, there is no selector, and the predicate is *ISPOINT*.

(*7) For a recent survey of attempts to define grammars for various classes of pictures, see Reference 85.

III.A. SUMMARY AND CONCLUSIONS

Experience in generating animated visual displays has shown that a computer with which one can interact directly and graphically, in real time, can be a powerful new medium for animation. The ability to see and alter a synthesized "movie" immediately is striking, for it cannot be achieved with traditional media and techniques. Interactive computer-mediated animation is the process of constructing movies by utilizing direct console commands, algorithms, sketches, and real-time actions. We have specified what is required for building interactive computer-mediated animation systems. We have implemented and used three such systems on the M.I.T. Lincoln Laboratory TX-2 computer. Thus the dissertation proves that the process is now technically (although perhaps not economically) feasible, and that it therefore deserves further research and development.

A process called picture-driven animation is a special kind of interactive computer-mediated animation that exploits the potentialities of direct graphical interaction. The animator may sketch and refine (1) static images to be used as components of individual frames of the movie, and (2) static and dynamic images that represent dynamic behavior, that is, movement and rhythm. These latter pictures, which eventually

drive algorithms to generate animated displays, depict data sequences called global descriptions of dynamics. Global dynamic descriptions abstract aspects of movement and rhythm which recur over extended intervals of time in particular animated displays. Picture changes that are continuous variations of value are expressed by path descriptions; picture changes that are recurring discrete choices of pictures, data, events, or actions are expressed by selection descriptions. Rhythm descriptions express patterns of the triggering, pacing, coordination, and synchronizing of picture change.

Each global dynamic description determines critical parameters of a sequence of frames. Thus, with a single sketch or action that generates or modifies such a representation, the animator can exercise precise dynamic control over an entire interval of the movie. More specifically, he does this by (1) sketching and editing graphically static representations of dynamic descriptions, and by (2) mimicking in real time desired dynamic behavior. He communicates with the system about a movie in an intuitively natural "language" of pictures and sketches.

Such a "language" of pictures and sketches has been implemented in several closed animation systems. They are closed in the sense that the animator can only sketch into the system and command the computer to aid the sketching process; he cannot in this command language describe

extensions to the language, that is, he cannot define new commands and their meanings. The dissertation lays the foundation for the design of a multi-purpose, open-ended, Animation and Picture Processing Language. APPL is a conversational language which accepts direct sketches, direct console commands, and algorithms that control interactive dynamic displays. It has several features that distinguish it from existing on-line graphics languages:

(1) The language can easily be augmented while remaining within the language; the system is, in a very powerful sense, extensible or open-ended.

(2) Strictly sequential flow of program control is abandoned in favor of the quasi-parallel execution of picture-transforming algorithms. This facilitates the specification and coordination of multiple strands of dynamic activity.

(3) A data structure called the aggregate, which is a generalization of all hierarchic ordered data representations, is used to model both complex pictures and global dynamic descriptions.

(4) An extensible class of picture descriptions called attributes, including one which abstracts the concept of picture "type," may be used by the animator to tailor a rich picture description capability to his own specifications.

(5) The animator's dynamics as expressed through a

stylus, push-buttons, and other devices is included as an integral component of animated system behavior. The stylus, for example, is modeled just as is any picture point, with the exception that its location cannot be set by a program. Thus the animator may write programs describing his own interaction with a movie as easily as he can algorithmically generate a movie.

The goal of the language design is plasticity in the representation of dynamic information and flexibility in the techniques and conventions with which the animator interacts with the system. It has been verified on paper that the language can be gracefully used to construct dynamic displays, to build system tools that aid the construction process, and to implement special-purpose interactive computer-mediated animation systems.

III.B. DISCUSSION AND SUGGESTIONS FOR FUTURE WORK

Specific observations and judgments shall be made on the strong and weak points of the TX-2 graphics environment for the implementation and use of ADAM, EVE, and GENESYS. In the hope that such documentation will assist future workers in the field, we shall augment the general published literature [illegible] specific to the application area of animation. (*1) We consider the following TX-2 subsystems:

(1) the computer itself, including its hierarchy of auxiliary storage;[89]

(2) the display processor;[90]

(3) APEX, the time-sharing monitor system;[91]

(4) the display executive subsystem of APEX;[92]

(5) the interrupt-processing executive subsystem of APEX;[92]

(6) the VITAL compiler-building system;[93] and,

(7) the LEAP Language for the Expression of Associative Procedures.[82]

Some remarks in this section suggest avenues towards the design of supporting subsystems which could better facilitate interactive computer-mediated animation. Some of these research questions are posed in more detail in III.B.4.

III.B.1. DIFFICULTIES ENCOUNTERED IN IMPLEMENTING AND USING ADAM, EVE, AND GENESYS

Specific observations and judgments shall be made on the strong and weak points of the TX-2 graphics environment for the implementation and use of ADAM, EVE, and GENESYS. In the hope that such documentation will assist future workers in the field, we shall augment the general published literature on the TX-2 system[82,87-94] with remarks specific to the application area of animation. (*1) We consider the following TX-2 subsystems:

(1) the computer itself, including its hierarchy of auxiliary storage;[89]

(2) the display processor;[90]

(3) APEX, the time-sharing monitor system;[91]

(4) the display executive subsystem of APEX;[92]

(5) the interrupt-processing executive subsystem of APEX;[92]

(6) the VITAL compiler-building system;[93] and,

(7) the LEAP Language for the Expression of Associative Procedures.[82]

Some remarks in this section suggest avenues towards the design of supporting subsystems which could better facilitate interactive computer-mediated animation. Some of these research questions are posed in more detail in III.B.4.

The TX-2 and its storage hierarchy

The TX-2 is an experimental, graphically oriented facility, usually time-shared among up to nine users, but frequently available for dedicated use. The system has 164K of core, a Fastrand II drum for bulk and swapping storage, and three magnetic tape units for back-up storage.

Asset: There are five refreshed CRT's, two storage scopes, and a wide variety of input devices, including two stylus-tablet-comparator units, knobs (shaft-encoders), toggle switches, push buttons, and typewriters.

Asset: The hardware is prepared to add new devices, and so it was relatively easy to add our computer-controlled movie camera.

Asset: When running in dedicated mode, a user can occupy over 100K of core.

Liability: The Fastrand II is very slow as a swapping drum. Delays on the order of fractional to several seconds can easily be encountered in animation applications. Faster secondary memory is definitely required.

Consequence: If programs or data representing a movie exceed the capacity of real core, then appropriate swapping must occur concurrent to computation or playback. To avoid delays in which the system waits for swapping, it must carefully control the allocation of real core and the initiation of swapping. This can easily be done during the playback

of a sequence of frames, as we shall discuss below. It is difficult to achieve during the computation of the sequence. To facilitate these tasks, the animation system itself, and the programs and data which define movies, should be factorable into meaningful units.

The display processor

Displays are refreshed, from ring-structured files residing in main core, by a time-shared analog signal generator of points, lines, conic sections, and characters.

Asset: Points, lines, and conics are stored in a uniform homogeneous matrix representation.[94]

Asset: Hierarchic structures of "groups" of display items may be established. A group may be "used" as an item in other groups, and the origin of each "use" given a unique offset. Thus, instances of a single subpicture can easily be placed at several locations on the scope.

Liability: The same beam is used for refreshing both the static and the dynamic portion of the display.

Consequence: Excessive flicker results when pictures are complex or the system is under heavy time-shared use. (*2) Movies are more likely to be display-bound, which occurs when frames cannot be painted on the screen in 1/24th of a second.

Implication: We have variable-speed playback in order to view display-bound movies at a slower rate. We need mechanisms for factoring animation sequences into parallel dynamic

strands, in order to reduce the complexity of the display by viewing separately different parts of the action in a movie.

Asset: The hardware translates subpictures dynamically, through the group offset mechanism.

Liability: The hardware has not yet been augmented to associate a single homogeneous matrix with each use of a group, and to multiply it into each picture matrix before display.

Consequence: Rotation, scaling, windowing, and clipping must currently be done by software. This is either costly of playback time, if done during playback, or costly of movie computation time and storage space, if done while the movie is being calculated, and the resulting images are stored for subsequent playback.

Implication: Addition of the hardware matrix multiplication would enable it efficiently to carry out dynamic rotation, scaling, and arbitrary projective transformations.

Asset: Since a separate channel refreshes the scope, and since a user's display file is frozen in core, his picture remains visible even while his program's execution has been interrupted in time-sharing.

Liability: If the picture is a movie being played back, however, playback stops and the current frame is held on the scope.

Implication: A standard format for movie display files should be adopted, and the concept of holding the picture on the display extended so that playback could continue even though

other animation computations were interrupted. Given the format used by ADAM, EVE, and GENESYS, switching frames requires only disconnecting and reconnecting one link in the display file structure. Modifying the hardware so that the display processor could do this independently would not be difficult.

The time-sharing monitor

Asset: APEX provides services including resource allocation, scheduling, and protection for time-shared users. There are also commands with which they can manipulate files and allocate virtual core.

Liability: Although a user running in dedicated mode could be allowed to allocate directly that portion of real core unused by the monitor, this is not possible under APEX.

Consequence: Swap-bound movies are more likely.

Liability: The scheduling philosophy does not allow a single user in time-sharing to request short-term absolute priority, even if he is willing to pay the penalty later on.

Consequence: Movie playback can be arbitrarily interrupted in time-sharing. Only when input functions can be totally assumed by the interrupt executive can real-time monitoring of the animator's movements and sketches be guaranteed. Hence, although we have prepared animation systems and partially debugged them in time-sharing, we rarely animate with more than one or two other users on the machine.

Liability: TX-2 programs operating under APEX have a push-down stack of virtual memories. However, only 16 "books" are allowed on each "map", or level of virtual memory, only one file may be put in a map, and communication between maps is difficult and time-consuming.
Consequence: Partitioning, or factoring, of programs, data, and images is difficult to implement.

The display executive

Asset: This APEX subsystem provides mechanisms for initializing, naming, saving, and retrieving display files; for building display items, combining them into groups, and hierarchically structuring these groups; and for interrogating an existing file to discover its hierarchic structure.
Liability: The structuring and accessing mechanisms are quite limited. A picture that appears only once on the scope cannot be included in more than one group of pictures. Although the file is ring-structured, and each group is stored as a ring, the ordering of items within a group is inaccessible to the user, and items must be referenced by a pair of integers, a group ID and an item ID.
Consequence: Most of the picture structure cannot be kept in the display file; an additional data base is needed. Updating and maintaining consistency between dual pictorial data bases adds significantly to programming complexity.

Liability: A display file is limited in size to one APEX book, or 8K registers. Only one such file can be displayed or referenced at a time. The executive system provides no mechanisms for communication between files.

Consequence: Since movies can easily exceed 8K of picture data, we have been forced to write our own mechanisms for communicating between display files, for guarding against overflow, and for structuring movies as groups of files. Since time delays occur in switching from file to file, it is advisable to restructure them so that all constituent subpictures of a contiguous sequence of frames can be found in a single file. This too adds significantly to programming complexity.

The interrupt executive

Asset: This APEX subsystem provides mechanisms for requesting and releasing input services, and for specifying the conditions under which a program should be interrupted and given information about a device. This information consists of the name and the state of the device which caused the interrupt, and the states of other associated devices.

Asset: These services are provided continuously, regardless of whether or not a user program is running. Thus, they are capable of real-time monitoring of devices even under time-sharing.

Asset: One of the services provided is the laying of a trail of "ink", consisting of a thinned and smoothed sequence of

points, over the trajectory of a free-hand sketch.

Liability: Only 256 points may be entered before the pen "runs out of ink". This sometimes forces an animator to decompose a free-hand sketch in unnatural ways.

The compiler-building system

Asset: VITAL, a compiler-compiler in which language syntax is specified in Floyd Productions and language semantics is specified in Formal Semantic Language (FSL), allows the flexible construction and modification of compilers.

Liability: FSL is useful for the algebraic component of a language, but is of little help in defining the semantics of list and associative processing. Therefore most of these semantics are written in assembly language.

Liability: VITAL produces compilers which translate programs into non-relocatable, non-modular code.

Consequence: This feature, and corresponding aspects of APEX, make program partitioning difficult. Thus, even small changes to a system are time-consuming because large parts of the system must be recompiled.

Implication: A compiler-building system for an interactive graphics environment should produce incremental compilers or interpreters.

The high-level programming language

Asset: LEAP is an ALGOL-like language augmented to facilitate

associative, display, and interrupt processing. It has been used for building several interactive graphics systems now operational on the TX-2.[33,36] Well-designed, high-level language forms allow one to structure and interrogate a data base, generate pictures, and monitor user input. The latter two tasks are achieved by LEAP's making calls on the APEX display and interrupt executive systems.

Liability: Little is understood about methods to achieve efficient processing of large associative data bases with conventional digital computers.

Liability: Data bases can be saved and recalled, but merger and communication is difficult because all names in a data base are local to that structure. More fundamentally, no symbolic communication between LEAP programs is possible. This restriction is due in part to VITAL.

Liability: Finally, we stress again the inconvenience of keeping pictures in both their LEAP and their display file representations.

Problems with the implementation of ADAM and EVE, and the solutions employed

ADAM and EVE are written in assembly code; the corresponding playback and filming programs are written in the algebraic subset of LEAP. In both cases, frequent calls are made on the time-sharing monitor, including its display and interrupt-processing subsets.

Problem: Currently, the movements of a single figure in ADAM can be computed at a rate of only 4 frames per second.
Solution: Effective real-time viewing does occur in playback, so the slow computing rate is of no consequence.
Problem: Only approximately 100 frames of picture information, (12 lines per frame, from a single figure in ADAM, or a pair of figures in EVE) can be stored in a display file.
Solution: Both ADAM and EVE fill sequentially a number of display files with adjacent frames. So that a long movie will not be swap-bound, the playback program reads the first n files into core, plays back the first one, and then begins to swap it out of core, replacing it with the (n+1)st while it plays back the second file, and so on. Despite its lack of direct control over real core, it can accomplish this by operating upon its virtual core. The success of this procedure in general depends upon the size of core memory, the size of each frame's representation in memory, the desired rate of playback, the maximum desired length of playback, and the speed of swapping when it is occurring simultaneously with the display.

Problems with the implementation of GENESYS, and the solutions employed

GENESYS and its playback and filming programs are written in LEAP with some assembly code interspersed. Little associative processing is used. Frequently calls on the monitor are made, particularly for display and interrupt services.

Unlike ADAM and EVE, GENESYS actually computes the movie during playback. ADAM must precompute complicated effects of several rotations and translations on each limb of the figure. In GENESYS, the algorithm which generates the movie is so simple that it is built into the display hardware. Each cel is stored in a unique group. Translation is achieved dynamically by changing the offset of the group. The selection among cels is achieved dynamically by switching cels in and out of that part of the structure which is made visible by the display processor. Provided that all the cels fit in a display file, these operations can be done during playback without loss of speed. If the additional matrix multiplication were added to the display as suggested above, then dynamic rotation and scaling could also be efficiently achieved without storing large files of images.

Problem: Since free-hand sketches are composed of long trails of points, flicker is a problem and movies easily become display-bound.

Solution: Movies can be factored by working on only a subset of the cel classes at a single time.

Problem: The cels required for an interesting animation sequence can easily exceed a display file.

Solution: Extra files must be generated according to the values of the selection descriptions, so that entire sequences of contiguous frames can be displayed without switching

between files during every frame can be avoided.

We briefly review areas for further work already proposed in the dissertation: I.B.10. noted that GENESYS should be augmented so that arbitrary projective transformations of cels can be applied continuously under the control of path descriptions, so that dynamic hierarchies can be established by defining "macros" of dynamic descriptions, and so that generalized-cels can be integrated smoothly into the system. As we stated in I.C.7., further research is needed on the problem of coordinating parallel actions in a movie. I.C.8. discussed the need for more in-depth experimentation, in which animators would work with and comment upon the suggested techniques for defining and refining dynamic descriptions.

We have constantly encountered and side-stepped the problem of constraint satisfaction. More research is needed both on the general problem and on specific instances that occur in animation. The demands of animation are severe, for the set of constraints can change in each frame.

Finally, further research should pursue the discussion of I.F. on "action-picture interpreters," on techniques for interpreting actions and pictures as representations of picture-transforming algorithms as well as representations of dynamic data.

III.B.2. WHAT TO DO NEXT IN PICTURE-DRIVEN ANIMATION

We briefly review areas for further work already proposed in the dissertation: I.B.10. noted that GENESYS should be augmented so that arbitrary projective transformations of cels can be applied continuously under the control of path descriptions, so that dynamic hierarchies can be established by defining "macros" of dynamic descriptions, and so that generalized-cels can be integrated smoothly into the system. As we stated in I.C.7., further research is needed on the problem of coordinating parallel actions in a movie. I.C.8. discussed the need for more in-depth experimentation, in which animators would work with and comment upon the suggested techniques for defining and refining dynamic descriptions.

We have constantly encountered and side-stepped the problem of constraint satisfaction. More research is needed both on the general problem and on specific instances that occur in animation. The demands of animation are severe, for the set of constraints can change in each frame.

Finally, further research should pursue the discussion of I.F. on "action-picture interpreters," on techniques for interpreting actions and pictures as representations of picture-transforming algorithms as well as representations of dynamic data.

III.B.3. THE IMPLEMENTATION AND FURTHER GENERALIZATION OF APPL

We hope that our outline of the design of APPL is a step towards a better understanding of the role formal language structure can play in the construction of interactive graphics systems. The next step is to implement APPL. In the course of this implementation, some areas of the outline need to be made more specific, for example, language syntax, block structure, and the choice of activity scheduling mechanisms.

The base language described in the dissertation is a relatively "pure" algorithmic language, containing very few "declarative" constructs whose only function is to allocate storage and to aid the processor in efficiently executing algorithms. Thus, there is currently no way to tell APPL that an aggregate could be implemented by a fixed-length storage structure, what the upper bound is, and what kind of error signal should result from a reference that is out-of-bounds. To preserve APPL's clarity, the declarative components introduced in further development should be kept as distinct as possible from the procedural components. We shall discuss in the next section how this separation in the language could be reflected in the way APPL is used operationally.

APPL's mechanisms for aggregation should be augmented so that they apply to commands as well as to numbers and pictures. APPL already contains one construct that is implicitly an aggregate of commands, the agenda. Further generalization

is needed to achieve the unification of the concepts of picture (an aggregate of pictures) and action (an aggregate of commands, that is, events or activities) that was discussed in I.F. It should be possible to combine groups of commands that generate pictures in the same way that one can combine the pictures themselves.

II.E.2. suggested that there should be further research on the difficult problems of introducing new picture types and of extending the meaning of picture attributes. One aspect of this would be the search for workable interpretations of the *TYPE* of a dynamic picture, for example, "*DEFINEATTRIBUTE TYPE of* LINE.0 *tobe* ROTATINGLINE;".

Finally, there is an issue of APPL's design that has thus far been ignored, but should be tackled so that an implementation can be as efficient as possible. Many animation sequences will be generated by several activities, or quasi-parallel computation strands. We will compute the sequence once, view the result, change one strand, compute again, and iterate in this fashion. APPL needs constructs with which the user and the system can cause images and other data resulting from computations to be stored and from then on retrieved rather than recomputed. This issue was mentioned in II.E.1. when we considered the evaluation of picture attributes.

The problem can be phrased more succintly by extending the concept of "time" developed in II.C.2. There are actually

three kinds of time, real time, movie time, and take time. Take time measures the number of trial computations, or takes, that must be executed in order to refine an animation sequence over a single interval of movie time. We say that a picture [property] is static if its value is fixed over an interval of movie time, and dynamic if it changes in regular relation to the advance of movie time. In the latter case, it may be represented by a dynamic picture [description]. The dynamic picture [description] may or may not change as take time progresses. The problem is therefore one of identifying static pictures [properties], and storing them so that they need not be recomputed, and of identifying dynamic pictures [properties] that have stabilized at some point in take time, and storing them so that they need not be recomputed. As take time advances, activities defining the movie can be individually deactivated until none are remaining and the movie is complete, stored as part of the pictorial data base.

III.B.4. HOW SUPPORTING SUBSYSTEMS, BOTH HARDWARE AND SOFTWARE, COULD BETTER FACILITATE INTERACTIVE COMPUTER-MEDIATED ANIMATION

Some hardware additions and modifications could make interactive computer-mediated animation more efficient and hence more economical. We have already suggested that:

(1) The display processor could be given the capacity for directly multiplying picture items by homogeneous matrices which represent projective transformations.

(2) The display processor could be given the capacity for playing back a movie without executing a user program, by cycling automatically through frames encoded in a specially-formatted display file.

Other suggestions follow:

(3) Algorithms implementing generalized-cels could be stored in special high-speed read-only memories.

(4) Dynamic descriptions could be stored and retrieved from a cyclic memory device such as a drum, and not occupy valuable core, since the data are accessed sequentially during movie computation or playback.

(5) In a similar vein, a display driven by a rotating magnetic disk might be ideally suited for playing back movies.[97-98] These have been used to drive low-cost graphic display terminals which operate on television principles.[97-99] (*3) The use of video tape as the movie storage and playback medium should also be considered.

(6) Finally, if conventional computer displays are employed to play back movies, then one should consider optically superimposing a static image from a storage tube with the dynamic image from a refreshed scope, thus enabling the latter to be more efficiently used.

We shall also raise two fundamental issues of computer design which are relevant to computer graphics generally, and which suggest important areas for research.

The computers of today have not been built to facilitate picture processing. Integers, fixed and floating point numbers, bytes, and bits are all primitives; pictures are not. Thus, commands to add two numbers, to shift a byte an integer number of places, and to take the logical product of two bit patterns are executed directly by current arithmetic units. Unfortunately, correspondingly basic operations for picture processing, such as superimposing two pictures, rotating a picture a number of units, and forming a line to connect two points, must all be expressed in terms of arithmetic and logical operations. Recalling the model for points developed in II.E.1., we can see that rotating a picture should result in an augmenting of its X and Y coordinates as well as its θ coordinate. In a computer designed for picture processing, such operations could be carried out automatically by the hardware.

Another fundamental problem arises in the implementation

of data bases which represent dynamic sequences, and the hardware and software mechanisms which interpret them. We noted in III.B.1. that maintaining dual pictorial data bases in current graphics systems is very awkward. What is needed is a single storage structure which can represent rich picture descriptions (aggregates and properties), yet can contain as a proper subset that minimal information needed to display the picture. A corresponding display processor is needed which can generate the picture directly from the proper subset of the total, unified, pictorial data base.

Next, we consider software subsystems and their relationship to effective interactive computer-mediated animation.

Again, we emphasize that the subsystems with which an animation system is implemented must facilitate the partitioning of programs, data, and large files of images.

If interactive animation is to succeed in time-sharing, then either the executive systems must assume all responsibility for generating real-time playback and for monitoring real-time input, or the monitor must allow an animation console to request absolute priority for brief intervals of time. Furthermore, these brief intervals cannot be assigned statistically; their occurrence must be predictable, granted upon demand, so that the animator can set himself to use the time effectively.

Finally, we suggest investigation of a new design philosophy for an interactive graphics system, one that

directly incorporates explicit roles for the animator, for the programmer, and for the computer. Suppose that the animator and the programmer could communicate concurrently with the system from respective and adjacent consoles. Recall the previous section's discussion of the algorithmic and declarative components of the language. While the animator defines and manipulates pictorial aggregates, the programmer, understanding the computational implications of the animator's activities, could furnish the system with information to help it allocate storage efficiently and make sensible implementation choices. Presumably the programmer would try to anticipate what information will be needed, although the system could also automatically direct queries to the programmer. The approach would be effective insofar as it allowed the animator to continue his work without interruptions and without concerning himself with details of the computer processing.

III.B.5. THE FEASIBILITY OF INTERACTIVE COMPUTER-MEDIATED ANIMATION

The dissertation establishes that interactive computer-mediated animation is currently feasible on an experimental basis. Its economic and practical viability, present and future, depends strongly upon three factors:

(1) The ability to solve certain technical problems, so that the resulting films will be commercially acceptable;

(2) The ability to reduce costs through economical system design, and through appropriate management of a wide variety of human and technical resources; and,

(3) The ability to find a common "language" for animators, artists, educators, and computer scientists.

(1) There are many technical problems in the production of commercially acceptable film. Some of them, such as synchronization of visual and sound tracks, are encountered in all animation processes. In this case, use of the computer may lead to solutions more elegant than those of the past. (*4) In picture-driven animation, sound track waveforms or rhythm descriptions obtained from them could be displayed as are other dynamic descriptions.

The problem of generating color film will soon be solved in principle, when color scopes are generally available, although accurate control of hue, saturation, and intensity will still be difficult. Even in black-and-white, picture

quality will vary unless we use high-performance scopes whose intensity, spot size, and line thickness can be precisely controlled.

We still need better descriptions and generative mechanisms for shape, line quality, and texture. Unless video playback techniques are adopted, it is difficult to view in real time intricate, rapidly changing shapes and textures. This is not necessarily a serious liability, since many animators strive for simple dynamic effects. (*5)

Finally, one can consider the possibility of animation in three dimensions, which raises the difficult problem of "hidden-line elimination." It appears unwise and unnecessary to deal with the complexities of drawing in true 3D.[100] A more viable approach is to work on a stack of parallel picture planes, and to obtain the illusion of depth through motion, through control of the size of objects, and through the elimination of hidden lines.

(2) An animation system has been designed economically if it allocates resources in correct proportion to the needs of the animator at a particular moment. Some work can easily be done in conventional time-sharing despite sluggish response; other work, as was noted in III.B.1. and III.B.4., requires absolute priority. The demand for response from an interactive graphics console is far more variable than that from a typewriter console. We need better models with which to

describe, estimate, and satisfy these demands, without the expense of totally occupying a large computer. There should also be research on the possibility of using a small computer augmented by secondary memory with rapid parallel information transfer.

We also need to find ways to organize the creative efforts of a wide variety of human and technical resources, such as animators, artists, educators, computer scientists, conventional digital computers, analog input devices, and video tape and associated television equipment.

(3) Successful computer animation requires finding a common "language" for individuals whose backgrounds and ways of thinking are exceedingly diverse. They must be better able to communicate effectively their respective insights into the animation medium and into design features for animation systems.

I hope that APPL may be a positive step towards the required language. Because APPL unifies the processes of defining dynamic displays and defining an animation system, much of the medium's mystery can be removed. Users should better understand that the computer animation medium, unlike any other animation media, can to some extent adjust to their styles and to their needs. They should also better understand what aspects of a system can be changed, and what aspects cannot be changed. It is our hope that the continuity

between display definition and system definition will foster the creative partnership between animators and computer scientists that we seek.

III.B.6. APPLICATIONS OF INTERACTIVE COMPUTER-MEDIATED ANIMATION

We note in the Introduction the demonstrated significance of the computer in aiding our understanding of the dynamics of complex natural phenomena through simulation and display. With interactive computer-mediated animation we can generate educational films that are primarily drawn rather than computed, for use in subject domains as diverse as the humanities, the arts, and the social sciences.

Various modes of educational filmmaking can be imagined. For example, a teacher could quickly generate with GENESYS a crude animation sequence, tailored to the needs of his individual class. Then this film, or one made with APPL, could be used as the basis for generating a family of movies with the same "cast of characters."

We still have much to learn about educational filmmaking. In a current paper,[101] Huggins and Entwisle eloquently describe the role of computer animation in fulfilling the great untapped potential of "iconic modes of communication and instruction," in producing "visual images that in their ability to communicate ideas are superior to traditional graphical images on paper or blackboard. Instead of static images, words, and mathematical symbols," they suggest, "we may create dynamic signs that move about and develop in self-explanatory ways to express abstract relations and concepts.

...A dynamic dimension is now available that requires the invention and development of new conventions and visual syntax appropriate to this new medium if it is to be fully used for communication and education." Huggins and Entwisle are pursuing these goals in the context of producing computer animated films for engineering education.[27]

Generating dynamic displays interactively should benefit those seeking to construct and film stimuli for experiments on visual perception, such as the perception of movement,[102-103] the perception of emotion in facial expressions,[104] and the perception of causality.[105]

Colby[106] is attempting to stimulate language development in nonspeaking mentally disturbed children by allowing them to interact with alphabet symbols and pictures of objects presented on a display, augmented by keyboard and voice communication. He is interested in introducing motion into these images.[107]

The stick figure work suggests several extensions and applications. There is currently much interest in analyzing human body movements for bio-engineering uses such as the construction of artificial limbs,[108] and experimentation with synthetic models at an interactive display might prove a useful adjunct to this work. One could attempt to animate the outlines of figures by reducing them to a "stick figure" with Blum's Medial Axis Transform,[109] animating the resulting

figures, and then taking the inverse transform to regenerate an outline. There is also current interest in using computer-animated stick figures as a synthetic tool in choreography.[110]

Finally, it is obvious that interactive computer-mediated animation is an excellent teaching aid for animators, because they obtain immediate feedback of the results of their work, and can attempt dynamic variations to an extent not possible with other animation media.

FOOTNOTES -- III.B.

(*1) Reference 87 is a 1969 review article on the TX-2 experience with interactive graphics in a time-shared environment. Issues of the Semiannual Reports of Reference 88 furnish a chronological account of recent developments at the TX-2.

(*2) Large display files and excessive flicker result in part because we have portrayed free-hand curves by closely spaced sequences of points, and not approximated them with conic sections. For a hardware and software approach to the generation of approximating segments, see Reference 95; other software approaches are described in Reference 62 and 96.

(*3) Reference 97 reviews three recent approaches to low-cost computer display terminals, including two which are based on television techniques.

(*4) See I.B.8., particularly Footnote 7, and also I.F.'s discussion of the Computer Image technique for automatically generating cartoon character mouth movements which are synchronized to the speech waveform.

(*5) Whereas the computer is ill-suited to the "Disney style" of animation, it is well-suited to the more modern "U.P.A. style." These styles are described by the British animator John Smith (Halas and Manvell,[2] p. 221) as follows:

> Generally speaking, the overall aesthetic attitude favoured in Disney's cartoon films is balanced in the following way. Complexity of detail is matched by complexity of movement.
>
> For example, when a figure walks, his tongue may flap, his hair flop, his ears wiggle, his eyelids shutter, his adam's-apple oscillate, his whole body down to his feet fluctuate with added movements. The rich, even sugary colouring and bulbous forms are matched by movements that resemble a bladder of water moving (if it could), floppily and sensuously. The sentimentality of mood is matched with coy, cute, sprightly, easy movement, and sadism by excessive distortion and squashing. In general, life is portrayed as it might move in a land where people had brains and bodies of soft sorbo rubber.

The matching of pictorial form and mood to design in U.P.A. films is, of course, very different. U.P.A.'s artists favour simplicity of form and simplicity of movement, the essence without the frills. Acid colours and sharp forms are matched by a movement resembling the way in which cane, glass and wire would move (if they could), springy, whippy, staccato. The wit and cynicism of these cartoons is acted out in slapstick of a high but blase kind. In general, this is life as it would be led in a land where people had brains and bodies of sheet metal. (Emphasis added)

REFERENCES

The following abbreviations are used:

FJCC nnnn	Proceedings of the nnnn Fall Joint Computer Conference
SJCC nnnn	Proceedings of the nnnn Spring Joint Computer Conference
CACM	Communications of the ACM
IFIP	Proceedings of the International Federation for Information Processing Global Conference

---

1. N McLAREN
   Quotation in animation exhibit in the Canadian Cinematique Pavilion at EXPO '68 in Montreal Canada National Film Board Canada

2. J HALAS R MANVELL
   The technique of film animation
   Hastings House New York 1959

3. R STEPHENSON
   Animation in the cinema
   A Zwemmer Limited London A S Barnes & Co New York 1967

4. E MARTIN
   Private communication

5. K C KNOWLTON
   A computer technique for producing animated movies
   SJCC 1964

6. K C KNOWLTON
   A computer technique for the production of animated movies
   Bell Telephone Laboratories Film

7. L L SUTRO
   A Model of Visual Space
   Biological Prototypes and Synthetic Systems, Volume 1
   Plenum Press New York 1962

8. The human use of computing machines
Bell Telephone Laboratories Symposium June 20-21 1966
K C KNOWLTON
Computer-Produced Movies
A M NOLL
Computer-Generated Three-Dimensional Movies
F W SINDEN
Synthetic Cinematography
W H HUGGINS
A Movie Language for Phasors and Signals

9. L MEZEI
Computers in design and communication: canadian graphics conference
Datamation August 1966

10. Conference on Computer Animation
Education Development Center Newton Mass July 17-18 1967

11. Proceedings of the 1967 UAIDE Annual Meeting

12. Computer Output as Art
1968 IEEE International Convention Record

13. FJCC 1968
A M NOLL
Computer Animation and the fourth dimension
J L SCHWARTZ E F TAYLOR
Computer displays in the teaching of physics
C CSURI J SHAFFER
Art, computers, and mathematics
J CITRON J H WHITNEY
CAMP--Computer Assisted movie production
N WINKLESS P HONORE
What good is a baby?
D D WEINER S E ANDERSON
A computer animation movie language for educational motion pictures

14. K C KNOWLTON
Computer-produced movies
Science Vol 150 26 November 1965

15. F W SINDEN
Force, mass, and motion
Bell Telephone Laboratories Film

16. MIT SCIENCE TEACHING CENTER
Scattering in one dimension
Film available on loan from the Atomic Energy Commission

17. C LEVINTHAL
Molecular model-building by computer
Scientific American Vol 214 No 6 June 1966

18. C LEVINTHAL
Computer construction and display of molecular models
Film

19. E E ZAJAC
Computer-made perspective movies as a scientific and communication tool
Comm ACM Vol 7 No 3 March 1964

20. E E ZAJAC
Two-gyro, gravity gradient attitude control system
Bell Telephone Laboratories film

21. S VANDERBEECK
Several animated films made with the aid of a computer

22. J H WHITNEY
Several animated films made with the aid of a computer

23. Design and the computer
Design Quarterly 66/67 Walker Art Center
Minneapolis Minn

24. A M NOLL
The digital computer as a creative medium
IEEE SPECTRUM October 1967

25. J REICHARDT
Cybernetic serendipity, the computer and the arts
Studio International London and New York 1968

26. J NOLAN L YARBROUGH
An on-line computer drawing and animation system
IFIP August 1968

27. W H HUGGINS D R ENTWISLE
Exploratory studies of films for engineering education
Department of Electrical Engineering The Johns Hopkins University Report to U S Office of Education
September 1968

28. T MIURA J IWATA J TSUDA
An application of hybrid curve generation-cartoon animation by electronic computers
SJCC 1967

29. L HARRISON W W JACQUISH
Private communications

30. R M BAECKER
Picture-driven Animation
SJCC 1969

31. A GUZMAN
Computer recognition of three-dimensional objects in a visual scene
Ph.D Dissertation Department of Electrical Engineering
M I T January 1969

32. S A COONS
Computer graphics and innovative engineering design; the super-sculptor
Datamation May 1966

33. W R SUTHERLAND
Computer assistance in the layout of integrated circuit masks
IEEE International Convention Digest 1968

34. J R LOURIE A M BONIN
Computer-controlled textile designing and weaving
IFIP August 1968

35. R M BAECKER
Planar Representations of Complex Graphs
M I T Lincoln Laboratory Technical Note 1967-1
6 February 1967

36. R M BAECKER
Experiments in on-line graphical debugging: the interrogation of complex data structures
Proceedings of the First Hawaii International Conference on System Sciences January 1968

37. V BUSH
As we may think
Atlantic Monthly July 1945

38. J C R LICKLIDER
Man-computer symbiosis
Trans IRE PGHFE HFE-1 4 1960

39. I E SUTHERLAND
Sketchpad: a man-machine graphical communication system
MIT Lincoln Laboratory Technical Report No 296
January 1963 Also SJCC 1963

40. Information
Scientific Issue September 1966

41. J C R LICKLIDER W E CLARK
On-line man-computer communication
SJCC 1963

42. J C R LICKLIDER
Man-computer partnership
International Science and Technology May 1965

43. G A MICHAEL
Pictures, computers, and input-output
D B Bobrow and J L Schwartz Editors Computers and the policy-making community Prentice-Hall 1968

44. D B PARKER
Solving design problems in graphical dialogue
Paper based on a chapter in On-line computer systems
W J Karplus Editor McGraw-Hill 1966

45. A VAN DAM
Computer driven displays and their use in man/machine interaction
Advances in computers Volume 7 1966

46. M S WOLFBERG
Computer displays
Survey paper and bibliography for EE 648 The Moore School of Electrical Engineering University of Pennsylvania 17 November 1966

47. Annual review of information science and technology
C Cuadra Editor Contains each year a survey article on Man-Machine Communication

48. I E SUTHERLAND
Computer graphics; ten unsolved problems
Datamation May 1966

49. J E WARD
Systems engineering problems in computer-driven CRT displays for man-machine communication
IEEE Transactions on Systems Science and Cybernetics June 1967

50. Adams Associates
The computer display review

51. H S CORBIN
A survey of CRT display consoles
Control Engineering December 1965

52. Display systems engineering
H R Luxenberg R L Kuehn Editors
McGraw-Hill New York 1968

53. M R DAVIS T O ELLIS
The rand tablet: a man-machine communication device
FJCC 1964

54. J F TEIXERA R P SALLEN
The sylvania data tablet
SJCC 1968

55. K H KONKLE
An analog comparator as a pseudo-light pen for computer displays
IEEE Transactions on Computers C-17 January 1968

56. L G ROBERTS
The lincoln wand
FJCC 1966

57. L GALLENSON
A graphic tablet display console for use under time-sharing
FJCC 1967

58. J E CURRY
A tablet input facility for an interactive graphics system
Proceedings of the 1969 International Joint Conference on Artificial Intelligence Washington D C May 1969

59. R LANDAU R PARDO
Private Communication

60. R ARNHEIM
Art and visual perception--a psychology of the Creative Eye
University of California Press Berkeley and Los Angeles 1967

61. S A COONS
Surfaces for computer-aided design of space forms
Project MAC Technical Report MAC-TR-41 MIT June 1967

62. T E JOHNSON
Arbitrarily shaped space curves for computer-aided design
MIT 1966 Summer Session Course on Computer-Aided Design
1-12 August 1966

63. M EDEN
Handwriting and pattern recognition
IRE Transactions on Information Theory Volume IT-8
Number 2 February 1962

64. P MERMELSTEIN M EDEN
Experiments on computer recognition of connected handwritten words
Information and Control Volume 7 Number 2 June 1964

65. J C SHAW
JOSS: Experience with an experimental computing service for users at remote typewriter consoles
Proceedings of the IBM Scientific Computing Symposium on Man-Machine Communication May 3-5 1965

66. R K MOORE W MAIN
Interactive languages: design criteria and a proposal
FJCC 1968

67. J WEIZENBAUM
On-line user languages
Nordisk Tidskrift for Informations-Behandlung (BIT)
Bind 6 Hefte Nr 1 1966

68. J C R LICKLIDER
Languages for specialization and application of prepared procedures
Proceedings of the 1965 Congress of Information System Sciences

69. L G ROBERTS
A graphical service system with variable syntax
CACM Vol 9 No 3 March 1966

70. K C KNOWLTON W H HUGGINS
Some thoughts on programming languages for computer animation
Unpublished paper

71. T E CHEATHAM JR A FISCHER P JORRAND
On the basis for ELF--an extensible language facility
FJCC 1968

72. D G BOBROW J WEIZENBAUM
List processing and extension of language facility by embedding
IEEE Transactions on Electronic Computers Volume EC-13
Number 4 August 1964

73. E L THOMAS
The storing and reuse of real-time graphical inputs
M S Thesis Dept of Electrical Engineering MIT June 1969

74. M M JONES
Incremental simulation on a time-shared computer
Project MAC Technical Report MAC-TR-48 (THESIS)
MIT January 1968

75. O J DAHL K NYGAARD
SIMULA--an ALGOL-based simulation language
CACM Volume 9 Number 9 September 1966

76. M D'IMPERIO
Data structures and their representation in storage: Parts I and II
Part I NSA Technical Journal Volume IX Number 3 1964 unclassified
Part II NSA Technical Journal Volume IX Number 4 1964 unclassified

77. B RAPHAEL D G BOBROW L FEIN J W YOUNG
A brief survey of computer languages for symbolic and algebraic manipulation
Paper in D G BOBROW Editor Symbol manipulation languages and techniques North-Holland Amsterdam 1968
also reprinted in IEEE Computer Group News Volume 1 Nos 5 and 6 September and November 1967

78. R M BALZER
Dataless programming
FJCC 1967

79. N WIRTH
On certain basic concepts of programming languages
Technical Report No CS 65 Computer Science Department
Stanford University 1 May 1967

80. R A KIRSCH
Computer interpretation of English text and picture patterns
IEEE-EC August 1964

81. D L LONDE R F SIMMONS
Namer: a pattern-recognition system for generating sentences about relations between line drawings
Proceedings of the 1965 ACM National Conference

82. J A FELDMAN
Aspects of associative processing
MIT Lincoln Laboratory Technical Note 1965-13 April 1965

P D ROVNER J A FELDMAN
An associative processing system for conventional digital computers
MIT Lincoln Laboratory Technical Note 1967-19 April 1967

P D ROVNER
The LEAP users manual
MIT Lincoln Laboratory Technical Note 1968-40 To be released

P D ROVNER J A FELDMAN
The LEAP language and data structure
IFIP August 1968

83. P J LANDIN
The mechanical evaluation of expressions
The Computer Journal Vol 6 No 4 January 1964

84. T A STANDISH
A data definition facility for programming languages
PhD Dissertation Carnegie-Mellon University May 18 1967

85. W F MILLER A C SHAW
Linguistic methods in picture processing--a survey
FJCC 1968

86. J V GARWICK
GPL, a truly general purpose language
Comm of the ACM Vol 11 No 9 September 1968

J V GARWICK J R BELL L D KRIDER
The GPL language: user's manual and formal description

87. W R SUTHERLAND J W FORGIE M V MORELLO
Graphics in time-sharing: a summary of the TX-2 experience
SJCC 1969

88. Graphics
Semiannual Technical Summary Reports to the Advanced Research Projects Agency
Submitted by MIT Lincoln Laboratory in recent years

89. W A CLARK et al
The lincoln TX-2 computer
Proceedings of the Western Joint Computer Conference
February 1957

90. T E JOHNSON
Analog generator for real-time display of curves
MIT Lincoln Laboratory Technical Report No 398 July 1965

L G ROBERTS
Conic display generator using multiplying digital-analog converters
IEEE Transactions on Electronic Computers EC-16 June 1967

H BLATT
Conic display generator using multiplying digital-analog decoders
FJCC 1967

91. J W FORGIE
A time- and memory-sharing executive program for quick-response on-line applications
FJCC 1965

92. L G ROBERTS M V MORELLO
MIT Lincoln Laboratory Group 23 Internal Memos on the Display and Interrupt Executive Systems

93. J A FELDMAN
A formal semantics for computer languages and its application in a compiler-compiler
CACM Vol 9 No 1 January 1966

L F MONDSHEIN
VITAL compiler-compiler system reference manual
MIT Lincoln Laboratory Technical Note 1967-12
February 1967

94. L G ROBERTS
Homogeneous matrix representation and manipulation of N-dimensional constructs
May 1965 Included in Reference 50
See also the explanation of homogeneous matrices in the introductory survey article in Reference 50

95. M L DERTOUZOS H L GRAHAM
A parametric graphical display technique for on-line use
FJCC 1966

96. H F LEDGARD
The manipulation of approximating functions on a graphical display facility
M S Thesis Department of Electrical Engineering MIT
September 1965

97. G A ROSE
Computer graphics communication systems
IFIP August 1968

98. Magnetic disc + TV monitors = low cost graphic display terminals
Information Display January/February 1968

99. U F GRONEMANN
Operation Manual for the CODITCON: A TV scan-conversion and image-storage system for computer displays
Electronic Systems Laboratory Report ESL-R-284 MIT
July 1966

100. T E JOHNSON
Sketchpad III--a computer program for drawing in three dimensions
SJCC 1963

101. W H HUGGINS D R ENTWISLE
Computer animation for the academic community

102. D C BEARDSLEE M WERTHEIMER
Readings in perception
Van Nostrand Princeton 1958 See especially Part III.D. "Perception of Successive and Changing Stimuli"

103. S S STEVENS
Handbook of experimental psychology
John Wile and Sons New York 1951 See especially Chapter 32 "Time Perception" by H WOODROW

104. R W WOODWORTH H SCHLOSBERG
Experimental psychology
Henry Holt New York 1954 See especially Chapter 5 "Expressive Movements"

105. A MICHOTTE
The perception of causality
Basic Books New York 1963

106. K M COLBY
Computer-aided language development in nonspeaking mentally disturbed children
Technical Report No CS 85 Computer Science Dept
Stanford University December 15 1967

107. K M COLBY
Private Communication

108. R CONTINI H GAGE R DRILLIS
Human gait characteristics

D S McKENZIE
Knee controls for artificial legs
Proceedings of a Symposium on Biomechanics and related bio-engineering topics R M KENEDI Editor Permagon Press New York 1964

109. H BLUM
A transformation for extracting new descriptors of shape
Symposium on Models for the Perception of Speech and Visual Form Boston Mass 11-14 November 1964

110. A M NOLL
Choreography and computers
Dance Magazine January 1967

A HUTCHINSON
A reply
Dance Magazine January 1967

APPENDIX O:

MULTIPLE-CHOICE EXAMINATION

by Miss Patricia Lewis and Mr. Ronald Baecker

For the benefit of academically-oriented readers, a short multiple-choice quiz is appended.

SELECT THE BEST ANSWER:

(1) ADAM does not have a head because:
   (a) very small ellipses are distorted by the TX-2 scope;
   (b) the computer dropped a bit;
   (c) in reality, his name is Ichabod Crane;
   (d) just when the head was to be included, the size of the program reached a file boundary;
   (e) the answer may not be ascertained from the dissertation.

(2) EVE is:
   (a) a snake;
   (b) the original swinger;
   (c) a serpent;
   (d) a multiple personality;
   (e) a collection of rubber-band lines.

(3) A crocodiless is:
   (a) a female alligator;
   (b) a male alligator;
   (c) a typographical error;
   (d) a female crocodile;
   (e) a female caiman;
   (f) a mythical beast.

(4) Computers are valuable in animation because:
   (a) they think;
   (b) they cannot think;
   (c) they are sorcerers' apprentices;
   (d) none of the above;
   (e) none of the below;
   (f) all of the below;
   (g) none of the above.

(5) A reason for inviting artists and animators, unfamiliar with computer graphics, to sketch into GENESYS is:
   (a) because Mr. Baecker cannot draw;
   (b) because they are very enthusiastic;
   (c) because they are often young and beautiful women;
   (d) because they are the clients for whom this work is intended.

BIOGRAPHICAL NOTE

Ronald Michael Baecker

Ronald Baecker was born in Kenosha, Wisconsin, on October 7, 1942. He attended Taylor Allderdice High School in Pittsburgh, Pennsylvania, and then received an S.B. degree in Physics with honors from MIT in 1963, and an M.S. degree in Electrical Engineering from MIT in 1964. Throughout his undergraduate years, he held a National Merit Scholarship; while working on his M.S., he was a National Science Foundation Fellow. During the academic year 1964-1965, he was a Deutscher Akademischer Austauschdienst Dankstipendiat, studying mathematics at the University of Heidelberg, Germany.

Since returning to MIT from Germany, he has pursued the Ph.D. degree in Computer Science in the Electrical Engineering Department. This degree will be officially granted in June of 1969. During his doctorial studies, he has assisted in teaching subjects including automatic computation, applied modern algebra, and information theory. He has also been a research assistant at Project MAC, and a summer employee and consultant at Lincoln Laboratory.

He is currently a Commissioned Officer in the Corps of the U.S. Public Health Service, serving at the Division of Computer Research and Technology at the National Institutes of Health at Bethesda, Maryland. His interests include computer graphics and interactive systems, animation, programming languages, debugging, graph theory, applications of computers to education, and the effects of computers on society.

He has recently published: Planar Representations of Complex Graphs, M.I.T. Lincoln Laboratory Technical Note 1967-1, 6 February 1967; Experiments in on-line graphical debugging: the interrogation of complex data structures, Proceedings of the First Hawaii International Conference on System Sciences, January, 1968; Picture-driven Animation, Proceedings of the Spring Joint Computer Conference, 1969.

CS-TR Scanning Project
Document Control Form

Date : 2/2/96

Report # LCS-TR-61

Each of the following should be identified by a checkmark:
Originating Department:

☐ Artificial Intellegence Laboratory (AI)
☒ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 350 (357-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☐ Single-sided or
☒ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☒ Unknown ☐ Other:_____

Check each if included with document:
☒ DOD Form ☒ Funding Agent Form ☒ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages (by page number): _____

Photographs/Tonal Material (by page number): 41-45, 49-51, 84, 104-106, 138, 140, 144, 152, 160, 163, 165, 167, 169, 171, 173,

Other (note description/page number):

Description :	Page Number:
IMAGE MAP	(1-350)
UNNUMBERED TITLE PAGE	2-350
SCANCONTROL, COVER, FUNDING AGENT, DOD, TRGT'S (3)	(351-357)

Scanning Agent Signoff:
Date Received: 2/2/96 Date Scanned: 2/2/96 Date Returned: 2/9/96

Scanning Agent Signature: Michael N. Cook

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Massachusetts Institute of Technology
Project MAC

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP
None

3. REPORT TITLE

Interactive Computer-Mediated Animation

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Ph.D. Thesis, Department of Electrical Engineering, June 1969

5. AUTHOR(S) *(Last name, first name, initial)*

Baecker, Ronald M.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 1969	352	110

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
Office of Naval Research, Nonr-4102(01) b. PROJECT NO. NR-048-189 c. RR 003-09-01 d.	MAC-TR-61 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. AVAILABILITY/LIMITATION NOTICES

This document has been approved for public release and sale;
its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
None	Advanced Research Projects Agency 3D-200 Pentagon Washington, D.C. 20301

13. ABSTRACT The use of interactive computer graphics in the construction of animated visual displays is investigated. In interactive computer-mediated animation, movies are formed from direct console commands, algorithms, free-hand sketches, and real-time actions (such as mimicking a movement or rhythm with a stylus or a push-button). The resulting movies can be immediately viewed and altered.

In picture-driven animation, the animator may sketch and refine (1) static images to be used as components of individual frames of the movie, and (2) static and dynamic images that represent movement and rhythm. These latter pictures drive algorithms to generate dynamic displays. Since each such picture determines critical parameters of a sequence of frames, a single sketch or action controls the dynamic behavior of an entire interval of the movie.

The dissertation also outlines the design of a multi-purpose, open-ended, interactive Animation and Picture Processing Language. APPL is a conversational language which accepts free-hand sketches, real-time actions, and algorithms that control interactive dynamic displays.

14. KEY WORDS

Computers
Computer animation
Computer graphics
Interactive graphics
Machine-aided cognition
Multiple-access computers
On-line computers
Real-time computers
Time-sharing
Time-shared computers

DD FORM 1 NOV 65 1473 (M.I.T.)

UNCLASSIFIED
Security Classification

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned

Date: 2/2/1996

M.I.T. Libraries
Document Services