Proceedings
IBM Scientific Computing Symposium
Man-Machine Communication

Proceedings OF THE IBM Scientific Computing Symposium ON Man-Machine Communication

IBM®
DATA PROCESSING DIVISION
WHITE PLAINS, NEW YORK

This volume is one of a series of proceedings of scientific computing symposiums which have been held by IBM to advance and summarize the art of computer applications to the solution of scientific problems.

Statistics	320-1936
Large-Scale Problems in Physics	320-1937
Combinatorial Problems	320-1938
Control Theory and Applications	320-1939
Simulation Models and Gaming	320-1940
Man-Machine Communication	320-1941

Library of Congress Catalog Card Number: 66-19009
Manufactured in the United States of America

PREFACE

This book is based on the sixth in a series of meetings organized by IBM to provide an opportunity for information exchange among leaders in various fields of mathematical and scientific computer usage. As in the earlier meetings, our aim has been to bring together those people who have broad experience and who have made significant contributions on the use of digital computing equipment in their fields of specialty. Similar volumes are planned for meetings covering other topics of interest to the scientist, mathematician, and engineer.

The IBM Scientific Computing Symposium on Man-Machine Communication was held at the Thomas J. Watson Research Center in Yorktown Heights, New York, on May 3, 4, and 5, 1965. The symposium was organized into five half-day sessions covering scientific problem-solving, man-computer interface, languages and communication, new areas of application, and man-computer interaction in the laboratory. Papers were presented by leading members of the scientific community from universities, government, and industry. After each presentation, the floor was open for periods of informal discussions, some of which have been included in modified form in these proceedings.

This book could not have been published without the cooperation of the speakers who presented papers and made them available for publication. Special thanks are due the technical planning coordinators, whose intimate knowledge of the field was invaluable in planning and organizing the sessions.

We wish to thank the participants in this symposium for spending their time with us and for helping us get a closer view of the contemporary picture of the state of the art in their areas of specialization.

ROSTER

General Chairman

DESIO, R. W., Director of Scientific Marketing, Data Processing Division, IBM Corporation, White Plains, New York

Session Chairmen

SESSION I

COURANT, R., Professor Emeritus, Courant Institute of Mathematical Sciences, New York University, New York, New York

SESSION II

DAVID, E. E., JR., Executive Director, Research, Communications Systems Division, Bell Telephone Laboratories, Incorporated, Murray Hill, New Jersey

SESSION III

BACKUS, J. W., Visiting Mackay Lecturer and IBM Fellow, Electrical Engineering, University of California, Berkeley, California

SESSION IV

LICKLIDER, J. C. R., Consultant to the Director of Research, Thomas J. Watson Research Center, IBM Corporation, Yorktown Heights, New York

SESSION V

HAMMING, R. W., Head, Numerical Methods, Research Department, Bell Telephone Laboratories, Incorporated, Murray Hill, New Jersey

Technical Advisers and Planning Coordinators

GOLDSTINE, H. H., Director of Scientific Development, Data Processing Division, IBM Corporation, White Plains, New York

LICKLIDER, J. C. R., Consultant to the Director of Research, Thomas J. Watson Research Center, IBM Corporation, Yorktown Heights, New York

PALL, G. A., Administrator, Scientific Programs, Data Processing Division, IBM Corporation, White Plains, New York

Symposium Program Manager

TARNOFF, N. H., Administrator, Scientific Programs, Data Processing Division, IBM Corporation, White Plains, New York

Session Coordinators

SESSION I

BRENNER, J. R., Systems Engineer, IBM Corporation, Philadelphia, Pennsylvania

SESSION II

DOTY, J. G., Systems Engineer, IBM Corporation, Houston, Texas

SESSION III

JOHNSON, C. W., Systems Engineer, IBM Corporation, Minneapolis, Minnesota

SESSION IV

O'DELL, L. M., Systems Engineer, IBM Corporation, Washington, D. C.

SESSION V

RUUD, R., Systems Engineer, IBM Corporation, Los Angeles, California

Symposium Program Coordinators (IBM)

DIVISION HEADQUARTERS

HOLSTEIN, D., Manager, Scientific Marketing Programs, Data Processing Division, IBM Corporation, White Plains, New York

EASTERN REGION

GOLDEN, J. T., Scientific Marketing Manager, IBM Corporation, New York, New York

FEDERAL REGION

LECHNER, H. D., Scientific Marketing Manager, IBM Corporation, Washington, D. C.

MIDWESTERN REGION

NEIMAN, P. B., Scientific Marketing Manager, IBM Corporation, Chicago, Illinois

WESTERN REGION

ORENSTEEN, R. B., Scientific Marketing Manager, IBM Corporation, Los Angeles, California

Participants

[AFFILIATIONS AT TIME OF SYMPOSIUM]

ABU-GHEIDA, O. M., IBM Corporation,
Kingston, New York

ANDERSON, R. H., Professor of Education, Graduate School of Education,
Harvard University, Cambridge, Massachusetts

ANDREWS, F. C., Director, Statistical Laboratory and Computing Center,
University of Oregon, Eugene, Oregon

ARNOLD, L. G., Administrator, Scientific Computing Center,
Apparatus and Optical Division, Eastman Kodak Company,
Rochester, New York

ARNOW, J. A., Lincoln Laboratory, Massachusetts Institute of Technology,
Lexington, Massachusetts

AUBUCHON, R., Section Supervisor, Computer Sciences,
McDonnell Automation Center, St. Louis, Missouri

BACKUS, J. W., Visiting Mackay Lecturer and IBM Fellow,
Electrical Engineering, University of California, Berkeley, California

BALL, W. E., Associate Professor, School of Engineering,
Washington University, St. Louis, Missouri

BARNHARD, H. J., Professor and Chairman, Department of Radiology,
School of Medicine, University of Arkansas, Little Rock, Arkansas

BATSON, A. P., Computer-Science Center, University of Virginia,
Charlottesville, Virginia

BAYLES, R. U., Systems Research and Development Center,
IBM Corporation, Cambridge, Massachusetts

BENNETT, C. A., Manager, Mathematics Department,
Pacific Northwest Laboratories, Division of Battelle Memorial Institute,
Richland, Washington

BENNETT, J. M., Professor of Physics (Electronic Computing),
Basser Computing Department, School of Physics,
University of Sydney, Sydney, Australia

BEUTLER, J. A., Manager, Digital Analysis and Computation,
Knolls Atomic Power Laboratory, General Electric Company,
Schenectady, New York

BILO, S. J., Manager—Technical Computing, Vertol Division,
The Boeing Company, Morton, Pennsylvania

BLAIR, F. W., Thomas J. Watson Research Center,
IBM Corporation, Yorktown Heights, New York

BLANKINSHIP, W. A., National Security Agency,
Washington, D. C.

BLOSE, W. F., Manager, Biomathematics Research Facility, Biomathematics, Baylor University College of Medicine, Houston, Texas

BODNAR, S. J., Manager, Process Engineering Section, Research and Development Department, Texas-U. S. Chemical Company, Port Neches, Texas

BOOKSTON, J. M., Senior Research Mathematician, Research Laboratories, General Motors Corporation, Warren, Michigan

BOSSERT, W., Harvard University, Cambridge, Massachusetts

BRAMLEY, M., Program Coordinator, Data Processing Department, Consolidated Edison Company of New York, Incorporated, New York, New York

BRATHOVDE, J. R., Professor and Director, Computer Center, State University of New York, Binghamton, New York

BRAUN, G. W., Chief Scientist, Pacific Missile Range, Point Mugu, California

BRENNER, J. R., Systems Engineer, IBM Corporation, Philadelphia, Pennsylvania

BROMBERG, H., Senior Consultant, Arms Division, CEIR, Incorporated, Arlington, Virginia

BROUGH, H. W., Supervisor, Mathematical Systems Section, Standards and Techniques Division, Shell Oil Company, New York, New York

BRYANT, J. H., Federal Systems Division, IBM Corporation, Bethesda, Maryland

BULL, F. W., Engineering Experiment Station, Virginia Polytechnic Institute, Blacksburg, Virginia

BURINGTON, R. S., Chief Mathematician, Bureau of Naval Weapons, Code R-14, Department of the Navy, Washington, D. C.

BURKE, R. L., Manager, Data Processing, Houston Data Service Center, Shell Oil Company, Houston, Texas

CAMERON, S. H., Scientific Adviser, Computer Sciences, IIT Research Institute, Chicago, Illinois

CARTER, W. L., Associate Dean of Faculties and Officer for Academic Planning, University of Cincinnati, Cincinnati, Ohio

CERVENKA, W. F., Central Research Laboratory, Socony Mobil Oil Company, Incorporated, Princeton, New Jersey

CHAMBERS, J. M., Statistics Department, Harvard University, Cambridge, Massachusetts

CHANG, H. W., Thomas J. Watson Research Center, IBM Corporation, Yorktown Heights, New York

CHASEN, S. H., Head, Man-Computer Systems Program, Research Laboratory, Lockheed-Georgia Company, Atlanta, Georgia

CLARKE, A., JR., Director, Computing Center, Fordham University, New York, New York

COBAS, A., Associate Director—Nuclear Center, University of Puerto Rico, San Juan, Puerto Rico

COLLINS, W. H., Principal Research and Development Engineer, Engineering Systems Division, Bureau of Public Roads, U. S. Department of Commerce, Washington, D. C.

COONS, S. A., Associate Professor, Mechanical Engineering Department, Massachusetts Institute of Technology, Cambridge, Massachusetts

COURANT, R., Professor Emeritus, Courant Institute of Mathematical Sciences, New York University, New York, New York

CRISS, D. E., Associate Dean of Faculty and Director of Computer Center, Rose Polytechnic Institute, Terre Haute, Indiana

CULLER, G. J., Director, Computer Center, University of California, Santa Barbara, California

CURTIS, J. R., Director of Operations Research, Operations Research Division, Scott Paper Company, Philadelphia, Pennsylvania

DAVID, E. E. JR., Executive Director, Research, Communications Systems Division, Bell Telephone Laboratories, Incorporated, Murray Hill, New Jersey

DAVIS, W. B., Assistant Treasurer, Methods Research, Bankers Trust Company, New York, New York

DEGAN, J. W., Associate Technical Director, Information Systems Laboratories, The MITRE Corporation, Bedford, Massachusetts

DENES, J. E., Head, Programming Division, Applied Mathematics Department, Brookhaven National Laboratory, Upton, New York

DENKER, G. R., Department Head, Production Engineering, Procter & Gamble Company, Cincinnati, Ohio

DESIO, R. W., Director of Scientific Marketing, Data Processing Division, IBM Corporation, White Plains, New York

DEVERE, G. J., Research Laboratories, General Motors Corporation, Warren, Michigan

DI GRI, V. J., Thomas J. Watson Research Center, IBM Corporation, Yorktown Heights, New York

DOBBS, G. H., Manager, Computer Center Department, Research and Technology Division, Systems Development Corporation, Santa Monica, California

DOBROWOLSKI, R. M., Director, Data Processing Center, General Precision, Incorporated, Wayne, New Jersey

DOLCH, J. P., Director of Research, University of Iowa,
Iowa City, Iowa

DORNHEIM, F. R., Coordinator, Technical Computing,
Electronics and Communications Department, Sinclair Oil Company,
New York, New York

DOTY, J. G., Systems Engineer, IBM Corporation,
Houston, Texas

DRAKE, A. E., Director of Computer Center, West Virginia University,
Morgantown, West Virginia

DREW, D., Computing Center Staff Analyst, Harvard University,
Cambridge, Massachusetts

ELDRIDGE, J., Systems and Data Processing, Merck & Company,
Rahway, New Jersey

ERLICK, D. E., Research Psychologist,
Wright-Patterson Air Force Base, Ohio

ESTABROOK, J. R., Mathematica,
Princeton, New Jersey

EVERSOLE, G. E., Research Specialist, North American Aviation, Incorporated,
El Segundo, California

FANO, R. M., Ford Professor of Engineering and Director of Project MAC,
Department of Electrical Engineering, Massachusetts Institute of Technology,
Cambridge, Massachusetts

FARBER, D., Supervisor, Systems Programming, Switching Systems Engineering
Division, Bell Telephone Laboratories, Incorporated, Holmdel, New Jersey

FENDER, D. H., Associate Professor, Biology and Electrical Engineering,
California Institute of Technology, Pasadena, California

FENICHEL, R. R., Teaching Fellow, Computation Laboratory,
Harvard University, Cambridge, Massachusetts

FENVES, S. J., Associate Professor of Civil Engineering,
Coordinated Science Laboratory, University of Illinois, Urbana, Illinois

FERRELL, J. K., Professor, Chemical Engineering,
North Carolina State University, Raleigh, North Carolina

FINN, J. D., Professor, Division of Social Sciences and Communication,
University of Southern California, Santa Monica, California

FISCHER, M. J., Harvard University,
Cambridge, Massachusetts

FOGEL, G. D., Chief of Computing Sciences,
Grumman Aircraft Engineering Corporation, Bethpage, New York

FREIBERGER, W., Professor of Applied Mathematics and Director of Computing
Laboratory, Brown University, Providence, Rhode Island

GALLIE, T. M., JR., Associate Professor of Mathematics,
Duke University, Durham, North Carolina

GARRETT, G. A., Director, Information Processing,
Lockheed Missiles & Space Company, Sunnyvale, California

GAUTNEY, G. E., JR., Supervisor, Scientific Computer Services,
Corning Glass Company, Corning, New York

GEAR, W. C., Professor, Department of Computer Science,
University of Illinois, Urbana, Illinois

GEISSLER, F. D., Director, Bureau of Electronic Data Processing,
Department of Highways, Harrisburg, Pennsylvania

GLEISSNER, G. H., Assistant Director, Computation and Analysis Laboratory,
U. S. Naval Weapons Laboratory, Dahlgren, Virginia

GOERTZEL, H. B., Chief, Information Processing and Display Division,
JCCRG, Joint Chiefs of Staff, Department of Defense, Washington, D. C.

GOLDEN, J. T., Scientific Marketing Manager, IBM Corporation,
New York, New York

GOLDEN, R. K., International Scientific Systems,
New York, New York

GOLDSTINE, H. H., Director of Scientific Development, Data Processing Division,
IBM Corporation, White Plains, New York

GOODMAN, S., Director, Computing Center,
Queens College, Flushing, New York

GOTT, A. H., Associate Head, Data Reduction Department,
Aerospace Corporation, San Bernardino, California

GRAHAM, J. W., Associate Professor of Mathematics and Director of Computing
Center, University of Waterloo, Waterloo, Ontario, Canada

GREIBACK, S., Harvard University,
Cambridge, Massachusetts

GRENANDER, U., Brown University,
Providence, Rhode Island

HAANSTRA, J. W., IBM Corporation,
Harrison, New York

HAGSTROM, S. A., Associate Director, Research Computing Center,
Indiana University, Bloomington, Indiana

HAMMING, R. W., Head, Numerical Methods, Research Department,
Bell Telephone Laboratories, Incorporated, Murray Hill, New Jersey

HANCOCK, J. E. H., Associate Professor in Chemistry,
Reed College, Portland, Oregon

HARSHBARGER, R. E., Chief Scientist,
National Military Command System Support Center, Washington, D. C.

HELLER, J., Director, Heights Academic Computing Facility,
New York University, University Heights, New York, New York

HERGET, P., Professor of Astronomy,
University of Cincinnati, Cincinnati, Ohio

HETHERINGTON, R., Director, Computation Center,
University of Kansas, Lawrence, Kansas

HOFFMAN, W., Director, Computing and Data Processing Center,
Wayne State University, Detroit, Michigan

HOLLINGSWORTH, J., Director, Computing Center,
Rensselaer Polytechnic Institute, Troy, New York

HOLSTEIN, D., Manager, Scientific Marketing Programs, Data Processing Division,
IBM Corporation, White Plains, New York

HUGHES, J., Shell Oil Company,
Houston, Texas

HUNTER, L., Director—EDL, Sylvania Electronics Systems—West,
Mountain View, California

HUSKEY, H. D., Professor, Electrical Engineering and Mathematics,
University of California, Berkeley, California

JACKS, E. L., Research Laboratories, General Motors Corporation,
Warren, Michigan

JOHNSON, C. A., Behavioral Sciences Staff Officer,
Andrews Air Force Base, Washington, D. C.

JOHNSON, C. W., Systems Engineer, IBM Corporation,
Minneapolis, Minnesota

KEHL, W. B., Director, Computing Center,
University of Pittsburgh, Pittsburgh, Pennsylvania

KELLER, R. F., Director, Computer Center,
University of Missouri, Columbia, Missouri

KILGOUR, F. G., Librarian, Yale Medical Library,
Yale University, New Haven, Connecticut

KOHMAN, V. E., Director, Computing Center, VTOL Systems Division,
Curtiss-Wright Corporation, Caldwell, New Jersey

KRENN, H., Branch Chief, Computer Laboratory, NASA,
George C. Marshall Space Flight Center, Huntsville, Alabama

LACHMAN, R., Professor of Psychology, State University of New York,
Buffalo, New York

LAIRD, D. T., Director, Computation Center, Pennsylvania State University,
University Park, Pennsylvania

LAMSON, B. G., Professor of Pathology, University of California,
Los Angeles, California

LANDE, J., Project Engineer,
Aerospace Corporation, Dallas, Texas

LECHNER, H. D., Scientific Marketing Manager, IBM Corporation,
Washington, D. C.

LEVINE, D. R., Harvard University,
Cambridge, Massachusetts

LEWIS, L. J., Professor of Electrical Engineering,
University of Washington, Seattle, Washington

LICKLIDER, J. C. R., Consultant to the Director of Research,
Thomas J. Watson Research Center, IBM Corporation,
Yorktown Heights, New York

LOGEMANN, G. W., Assistant Professor, New York University,
New York, New York

LONDON, R. L., Assistant Professor, Computer Sciences Department,
University of Wisconsin, Madison, Wisconsin

LOWENSCHUSS, O., Consulting Scientist, Missile Systems,
Raytheon Company, Bedford, Massachusetts

LUBIN, J. F., Director of Computing Activities,
University of Pennsylvania, Philadelphia, Pennsylvania

MAPLE, C., Director, Computation Center,
Iowa State University, Ames, Iowa

MARTELLOTTO, N. A., Supervisor, Electronic Switching Division,
Bell Telephone Laboratories, Incorporated, Holmdel, New Jersey

MASAITIS, C. M., Chief, Applied Mathematics Branch,
Aberdeen Proving Grounds, Maryland

MASSEY, L., JR., Harvard University,
Cambridge, Massachusetts

MAUDLIN, C. E., JR., Director, Computer Laboratories,
University of Oklahoma, Norman, Oklahoma

MCBRIEN, V. O., Chairman, Department of Mathematics,
Holy Cross College, Worcester, Massachusetts

MCCLUNG, L. N., Applied Physics Laboratory, Johns Hopkins University,
Silver Spring, Maryland

MCCONNELL, W. A., Director, Systems Research Office, Engineering Staff,
Ford Motor Company, Dearborn, Michigan

MCPHERSON, J. C., Systems Research Institute,
IBM Corporation, New York, New York

MEISLING, T. H., Assistant Executive Director, Engineering Sciences and Industrial
Development, Stanford Research Institute, Menlo Park, California

MIDDLETON, W. C., Senior Scientist, Douglas Aircraft Company, Incorporated, Santa Monica, California

MILLER, W. F., Professor of Computer Science, Stanford University, Stanford, California

MOORE, E. C., Graduate Dean, University of Massachusetts, Amherst, Massachusetts

MORISON, J. S., Manager, Computing Engineering, Missiles & Space Systems Division, Douglas Aircraft Company, Incorporated, Santa Monica, California

MORRISS, B. E., Assistant Deputy Director for NMCS, NMCS Directorate, Defense Communications Agency, Washington, D. C.

NAGAI, A. T., Research Engineer, Aerospace Group, The Boeing Company, Seattle, Washington

NAVARRO, S., Computing Center Director, University of Kentucky, Lexington, Kentucky

NEIMAN, P. B., Scientific Marketing Manager, IBM Corporation, Chicago, Illinois

NELSON, D. J., Director, Computing Center, University of Nebraska, Lincoln, Nebraska

NICHOLSON, G. E., JR., Chairman, Department of Statistics, University of North Carolina, Chapel Hill, North Carolina

O'DELL, L. M., Systems Engineer, IBM Corporation, Washington, D. C.

OETTINGER, A. G., Aiken Computation Laboratory, Harvard University, Cambridge, Massachusetts

OLSEN, W. D., Director, Computing Systems Planning, North American Aviation, Incorporated, El Segundo, California

ORENSTEEN, R. B., Scientific Marketing Manager, IBM Corporation, Los Angeles, California

OSER, H., Consultant to Chief, Applied Mathematics Division, National Bureau of Standards, Washington, D. C.

PALL, G. A., Administrator, Scientific Programs, Data Processing Division, IBM Corporation, White Plains, New York

PARKER, S. T., Director, Computing Center, Kansas State University, Manhattan, Kansas

PAULSEN, R. E., Principal Staff Engineer, Aerospace Division, Martin Marietta Corporation, Denver, Colorado

PAVKOVICH, J., Assistant Manager, Applied Mathematics and Computer Sciences, Varian Associates, Palo Alto, California

PEACOCK, J., Business Week, Boston, Massachusetts

PEPLER, R. D., Dunlap & Associates,
Darien, Connecticut

PERLIS, A. J., Professor, Carnegie Institute of Technology,
Pittsburgh, Pennsylvania

POPOVITCH, L., Manager, Computing Services, Process Plants Division,
Foster-Wheeler Corporation, Livingston, New Jersey

PROWSE, W. J., Harvard University,
Cambridge, Massachusetts

RADFORD, K. J., Director, Central Data Processing Service Bureau,
Ottawa, Ontario, Canada

RAMOS, M. M., Director, Computing Center,
Ministry of Mines and Hydrocarbons, Caracas, Venezuela

RANDELL, B., Thomas J. Watson Research Center,
IBM Corporation, Yorktown Heights, New York

RANSDELL, L. F., Langley Research Center,
NASA, Hampton, Virginia

REEVES, R. F., Director, Computer Center,
Ohio State University, Columbus, Ohio

REINTS, R. E., Supervisor, Applied Mathematics Section,
Engineering Research Division, Deere & Company, Moline, Illinois

ROBERTS, A. E., Department of Defense,
Washington, D. C.

ROBICHAUD, L. P. A., Directeur, Centre de Traitement de L'Information,
Universite Laval, Quebec, Quebec, Canada

ROBINSON, G. A., Associate Mathematician, Applied Mathematics Division,
Argonne National Laboratory, Argonne, Illinois

ROBINSON, J. A., Professor of Computer Science and Philosophy,
Rice University, Houston, Texas

ROBINSON, R. J., Director, Computing Center,
Marquette University, Milwaukee, Wisconsin

ROBISON, C. C., Manager, Technical Support, Military Airplane Division,
The Boeing Company, Wichita, Kansas

ROOTHAAN, C. C. J., Professor, University of Chicago,
Chicago, Illinois

ROSEVEAR, J. W., Staff Consultant, Section of Biochemistry,
Mayo Clinic, Rochester, Minnesota

ROTHENBERG, L., Head, Display and Presentation Systems Development Office,
U. S. Weather Bureau, Washington, D. C.

RUDAN, J. W., Director, Computing Center,
Cornell University, Ithaca, New York

RUUD, R., Systems Engineer, IBM Corporation,
Los Angeles, California

RUYLE, A., Mathematician, Computation Laboratory,
Harvard University, Cambridge, Massachusetts

SAMUEL, A., Thomas J. Watson Research Center,
IBM Corporation, Yorktown Heights, New York

SCHWENK, H. S., JR., Harvard University,
Cambridge, Massachusetts

SCOTT, D. B., Professor and Head of Department of Computing Science,
University of Alberta, Edmonton, Alberta, Canada

SELFRIDGE, R. G., Department of Mathematics,
University of Florida, Gainesville, Florida

SHAH, R., Harvard University,
Cambridge, Massachusetts

SHAW, J. C., Computer Sciences Department,
The RAND Corporation, Santa Monica, California

SHEPARD, R. B., University of Alabama Medical Center,
Birmingham, Alabama

SHERADEN, G. H., Manager, Scientific Computing, ARO, Incorporated,
Arnold Air Force Station, Tennessee

SHIMAMOTO, Y., Chairman, Applied Mathematics Department,
Brookhaven National Laboratory, Upton, New York

SHU, H., Research Associate, Research and Development Division,
Lord Manufacturing Company, Erie, Pennsylvania

SMITH, O. D., Supervisor, Computing Technology, Los Angeles Division,
North American Aviation, Incorporated, Los Angeles, California

SNYDER, J. N., Associate Head, Department of Computer Science,
University of Illinois, Urbana, Illinois

SOEHNGEN, H., Director, Computing Center,
Polytechnic Institute of Brooklyn, Brooklyn, New York

STARK, R. H., Associate Professor of Information Science,
Washington State University, Pullman, Washington

STARKWEATHER, J. A., Acting Director, Computing Center,
University of California, San Francisco, California

STOCKHAM, T. G., JR., Assistant Professor, Electrical Engineering Department,
Massachusetts Institute of Technology, Cambridge, Massachusetts

SUPPES, P., Director, Institute for Mathematical Studies in the Social Sciences,
Stanford University, Stanford, California

TARNOFF, N. H., Administrator, Scientific Programs, Data Processing Division,
IBM Corporation, White Plains, New York

TAUB, A. H., Director of Computer Center and Professor of Mathematics, University of California, Berkeley, California

TAYLOR, R. W., Deputy Director for Information Processing Techniques, Advanced Research Projects Agency, Office of the Secretary of Defense, Washington, D. C.

TAYYABKHAN, M. T., Supervisor, Systems Research, Socony Mobil Oil Company, Incorporated, Paulsboro, New Jersey

THOMPSON, G. T., Manager, Computing Center, United Technology Center, Division of United Aircraft Corporation, Sunnyvale, California

TISCHHAUSER, J. L., Manager, Computer Programming Department, Sandia Corporation, Albuquerque, New Mexico

TONGE, F. M., Director, Computer Facilities, University of California, Irvine, California

TUROFF, M., Research and Engineering Support Division, Institute for Defense Analyses, Arlington, Virginia

UNCAPHER, K. W., Engineer, The RAND Corporation, Santa Monica, California

VAUGHAN, H. E., Director, Electronic Switching System Center, Bell Telephone Laboratories, Incorporated, Holmdel, New Jersey

VAZQUEZ, A., Director, Centro Nacional de Calculo, Instituto Politecnico Nacional, Mexico, D. F., Mexico

VAZSONYI, A., Scientific Adviser, Management Planning, North American Aviation, Incorporated, El Segundo, California

VELEZ-OCON, C., Assistant Chief, Planning Department, Compania de Luz y Fuerza del Centro, S. A., Mexico, D. F., Mexico

VERRILL, W., Computing Center Director, Maine State Highway Department, Augusta, Maine

WADE, J. W., Research Manager, Applied Mathematics Division, E. I. du Pont de Nemours & Company, Incorporated, Aiken, South Carolina

WALDEN, J. M., Assistant Professor, Electrical Engineering, Oklahoma State University, Stillwater, Oklahoma

WALDEN, W. E., Director, University Computing Center, University of Omaha, Omaha, Nebraska

WALOWITZ, H. L., Manager, Computer Support Group, Graduate School of Business Administration, New York University, New York, New York

WARD, J. A., Staff Specialist (Computer Technology), Office of the Director of Defense Research and Engineering, Office of the Secretary of Defense, Washington, D. C.

WAY, F., III, Associate Director, Computing Center,
Case Institute of Technology, Cleveland, Ohio

WEINBERGER, E. B., Research Associate, Evaluation and Computation Division,
Gulf Research & Development Company, Pittsburgh, Pennsylvania

WEISSMAN, H. B., Director, Computer Program,
University of Illinois, Chicago, Illinois

WILLIAMS, L. H., Assistant Director of Computing Center and Assistant Professor
of Mathematics, Florida State University, Tallahassee, Florida

WILSON, H. B., Manager, Computer Communication Department,
Socony Mobil Oil Company, Incorporated, Princeton, New Jersey

WITHINGTON, F. G., Operations Research Section, Arthur D. Little, Incorporated,
Cambridge, Massachusetts

WITTENBORN, A. F., Vice President, Tracor, Incorporated,
Austin, Texas

WOODS, R. E., Corporate Technical Staff Consultant,
Sanders Associates, Incorporated, Nashua, New Hampshire

YNTEMA, D. B., Leader, Psychology Group, Lincoln Laboratory,
Massachusetts Institute of Technology, Lexington, Massachusetts

ZIMMERMAN, M. B., Assistant Director, Scientific and Engineering Evaluation
Division, Computer System Directorate, U. S. Aids Command,
Washington, D. C.

CONTENTS

SESSION I

Scientific Problem-Solving

1
Solving Problems with Long Run Times

CLEMENS C. J. ROOTHAAN
University of Chicago

When the first digital computers became available in the early 1950's, their use was primarily restricted to problems of the highest priority, which usually meant problems connected with national defense. Since the middle 1950's, however, computers have become so plentiful that their application to many other problems has now become commonplace. From this time on, I have involved myself heavily in the use of computers for solving theoretical problems in atomic and molecular physics. I have always tried to use computers to the limit of what they could yield, and in the process I have also become very interested in computers themselves. I'd like to share with you some of the many interesting things I have learned during the last ten years about the use and organization of computer facilities.

Let me first present to you briefly some background about the field of physics I am active in. In a sense, it all started in 1913 when Bohr proposed his radically new concept of how electrons move in the shells of atoms. According to Bohr's theory, the electrons move in orbits which can be calculated by applying Newton's equations of motion of classical mechanics, which had been established several hundred years before. However, out of all possible motions determined in this way, a much more limited, and usually discrete, set was actually permitted; the principle for making this selection was provided by the so-called quantum postulate.

The next milestone in the development of atomic theory was the year 1925, when Bohr's model, which was still rooted to some extent in classical mechanics, was replaced by a still more radical concept. This time, classical mechanics was completely swept away, and Newton's equations of motion were replaced by the now famous Schrödinger equation, or the mathematically equivalent matrix formulation of Heisenberg. In the formulation of Schrödinger, an electron orbit is replaced by a function in three-dimensional space, called a wave function; the Schrödinger equation is nothing

but the differential equation which such wave functions satisfy. The connection with Bohr's theory is provided by the fact that the wave functions have large values at and around the points traversed by the Bohr orbits, and much smaller values outside of these regions.

Strictly speaking, this sketch as outlined above applies to the motion of a single electron in a force field, the most important case being the Hydrogen atom. However, the extension to many-electron problems is a very natural process for the Schrödinger equation, whereas the Bohr model poses many ambiguities when this generalization is attempted. The Schrödinger equation, then, becomes a partial differential equation for a wave function in which the position coordinates of all the participating electrons are the arguments. To the best of our knowledge, a wave function which satisfies this equation correctly represents an electronic system in a well defined stationary state, to the extent that observable chemical and spectroscopic properties of such a system can be reliably and accurately calculated.

Unfortunately, although the Schrödinger equation provides the correct equation of motion for the electrons in atoms and molecules, its solution is a mathematical problem of staggering proportions. It is easy to see that exact numerical solutions are out of the question, and will probably remain so for all time. Dealing with an average molecule, or an atom at the upper end of the periodic table, we have roughly 100 electrons. This is still a long way from large organic molecules or solid state devices, where a very much larger number of electrons is involved. Continuing with the case of 100 electrons, the wave function contains 300 independent variables. Since the Schrödinger equation is not separable, if we assume about 100 points for each coordinate as a satisfactory mesh, we are led to a numerical tabulation of 10^{600} entries. This number is literally super-astronomical, since it exceeds by a very large factor the number of particles in the universe.

It is clear, then, that in order to arrive at practical results for many-electron systems, one has to be satisfied with approximate solutions of the Schrödinger equation. Such approximations are usually put forward intuitively by physicists, and their most important character deals with reducing the many-particle aspect of the problem as much as possible, without destroying all correspondence with physical reality. The most successful model of wide scope has been the factorization of the wave function into one-electron functions; this was introduced by Hartree and Fock for atoms, and by Hund and Mulliken for molecules. Of course, the correct wave function cannot be factored; what the model proposes is to find the best approximation of factored form. This requirement then yields for an N-electron problem N partial differential equations in 3 dimensions; in each such equation, the solutions of the other $N - 1$

equations appear in coupling terms, most of which include integrations. The physical interpretation of this model is closely akin to the Bohr concept for many-electron atoms: each electron moves in a force field of its own, which represents, besides the attraction of the nucleus, the repulsion by a negative-charge distribution calculated from the average positions of all the other electrons. This physical interpretation has been reflected in the name coined for this mathematical model, namely, the method of the self-consistent field.

Let me now review the history of actual computations in this special area of theoretical physics. In the case of atoms, one further simplification is provided by the central symmetry. Namely, if the one-electron functions are expressed in terms of spherical coordinates, the angular dependence of these functions is given by spherical harmonics, and only a radial function remains to be determined. Thus the coupled partial differential equations in three-dimensional space reduce to coupled ordinary differential equations in a single variable. These equations are quite manageable even for hand computations, and calculations of this type have been carried out since the early 1930's. However, the calculations are quite laborious, and without computers it would hardly be feasible to obtain the self-consistent field functions for all the desired or interesting atomic cases. The situation is quite different for molecules. Only for one very simple case was the self-consistent field function determined by hand computation, namely, for the Hydrogen molecule in 1938 by Coulson. For anything beyond this two-electron molecule, the aid of the computer is absolutely essential to obtain self-consistent field functions; during the last five years, they have been obtained for several dozen diatomic molecules, and calculations on triatomic molecules are in progress. To properly appreciate the need for computers, it is quite possible that such a calculation, for just one molecule in one particular state, would require 100-1,000 man-years if done by hand; with present computers, this result can be obtained in something like 10-100 hours.

The length and complexity of such calculations poses several interesting problems. It requires, first of all, that the mathematical procedure by which the end result is achieved be understood to much greater depth than was necessary for hand calculations. Namely, if one proceeds with a really complex calculation, one may find that at a particular stage in the computation a certain algorithm does not yield all the accuracy desired, or does not converge efficiently, etc. Since many of these occurrences depend on the actual numerical situation of the particular case at hand, effective means have to be designed to let the program make the necessary choices while the computation proceeds. One might hope that man-machine interaction would help this situation, letting the user make these choices from time to time at appropriate breakpoints. However,

in a molecular calculation lasting only several hours, such interaction would occur too often if all these choices were to be settled by man-machine interaction, and would make the total time span of the computation impractically large.

Another interesting aspect of this type of calculation is the extraordinary complexity of the arrays to be handled. Quantities with six indices or more are commonplace, and the ranges of these indices do not obey the simple rule that they fill a parallelepiped of grid points in multidimensional space. Also, often the same array is used several times, but the order in which the elements are to be acquired may be very different for subsequent passes. If such a problem is coded in any of the currently available compilers, the inefficient use of storage for such complex arrays, and the slow speed of execution caused by other than sequential manipulation of the elements, will render the computation prohibitively expensive, and often even impossible. Current compilers simply lack the power and flexibility necessary for these applications. I am furthermore of the opinion that even as compilers improve significantly, there will always be a need for assembly language coding for certain problems which tax the resources of a computer system to the limit. In this respect, I consider the prevailing pressure to write all user programs in compiler language a disservice to the computing public.

My remaining remarks will be concerned with the organization of a computer system as a service facility. Before the University of Chicago obtained its own IBM 7090 in October, 1962, I had used off-campus facilities, often hundreds of miles away from Chicago. My general experience during this period was that computation centers are organized to be of benefit to the large number of users who do not tax the computer to the limit. My frequent needs for somewhat unusual procedures were often frustrated by operating rules and systems conventions which prevented me from getting my work done efficiently. The computation center management usually found my requests for such procedures unjustified and inconvenient, and could not understand that these demands constituted a legitimate need, in order that an application of unusual complexity be accomplished efficiently. Aside from being wasteful of the scientist's time, and therefore very annoying to him, this is also very short-sighted from the management's point of view. For it is precisely these users who are most creative with computers, and their knowledge and experience are likely to be of considerable value for designing and implementing better systems and operating procedures, from which even the occasional or light user will benefit.

A similar situation prevails in the relationship between computer manufacturers and sophisticated users. In many instances, new hardware or software features are designed by engineers or scientists who lack the

experience of having designed and/or written truly complex applications programs. Feedback from the latter to the former is absolutely essential for good hardware and software design; if this does not exist, such a design can have flaws which are not apparent until a machine is "put through its paces." A typical example of such a failure can be seen if one follows the history of the data channels on the IBM 709–7090–7094. Data channels were, of course, conceived to attain overlap between input, output, and computing. However, it was at first not realized that the main program must be able to receive signals from the input-output operations after they have been triggered off; the channels on the 709 lacked this capability. Again, the 7090–7094 channels had most but not all of this required capability. Furthermore, when our systems staff at the University of Chicago attempted to use the full power of channels in a general-purpose tape handler, in which the channel chains its commands, it was found that for certain sequences of commands error recovery was impossible, since the exact command on which the error occurred was ambiguous. This state of affairs clearly demonstrated a lack of understanding on the part of the designers of the channels. It furthermore brings into focus the crucial importance of simultaneous hardware and software design of new computer systems. A global overview of programming systems architecture is simply not an adequate substitute for this, since it is often the details in software implementation which point up the flaws in hardware design.

When the University of Chicago established its own Computation Center, my persistent dissatisfaction with the quality of service in many other installations was probably the reason I became its director. In this capacity, I considered it my task to eliminate those organizational shortcomings which had so often plagued me as a user. Perhaps the single most important impediment to the usefulness of a computer is the job turnaround time. It is ironical that as computers became faster and more powerful, the turn-around time increased steadily. This has become such an impediment that hardware is now being built to give many users simultaneous rapid access to the computer. However, with current equipment vast improvements are possible. At our center, a unique IBM 7094–7040 system has recently been installed, and the necessary software to support this configuration is being implemented. This arrangement differs from the direct-couple system inasmuch as the 7094 remains entirely compatible with a stand-alone 7094. The 7040 functions as the input-output computer and handles all the necessary job scheduling; the 7094 continues to operate as a standard two-channel tape machine. When the software is fully implemented, we expect to achieve a job turn-around time of about $\frac{1}{2}$ hour, even during heavy work loads, and a few minutes when the traffic is somewhat less.

2

On Time-Sharing Systems, Design and Use

A. H. TAUB
University of California, Berkeley

1. Introduction

During the past decade computer hardware has been greatly improved. Machines have become faster, more reliable and have been provided with larger random-access memories at more reasonable costs. A major portion but not all of this improvement is accounted for by advances made in obtaining faster and more reliable computer components. Novel ideas in the logical organization of computer sub-units have also contributed to the improvement of computers and in some cases have necessitated a departure from the classical organization of The Institute for Advanced Study type of machine.

Thus the speed of arithmetic units has been increased in good part because novel arithmetic algorithms have been discovered, and hardware has been organized to implement these algorithms. Present-day arithmetic units are not only faster than they used to be but are faster relative to memory speeds. In the early days of computers, machines had a multiply time of 40 main memory accesses. Quite a few of today's computers can multiply in times less than five times the memory access time.

In order to redress the unbalance due to the unevenness in our progress in these two areas of computer development, machine designers have organized computers so that the need for memory accesses has been decreased. This has been accomplished by a variety of means including changes in memory-addressing schemes and the introduction of complicated instructions. Such implementation makes use of storage registers outside the main memory—and additional hardware—so that it is possible to carry out red-tape calculations outside of the main arithmetic unit and at the same time as that unit is otherwise engaged. By using additional storage outside of the main memory and by providing additional control units, various look-ahead, or look-behind, units have been devised to further decrease the number of main memory accesses and to increase the amount of work being done concurrently.

Thus present-day "advanced" computers are designed to gamble on the predictability in small or local parts of a scientific numerical calculation. The gamble is hopefully of the sort in which gains are made when it is successful, but losses are minimized or non-existent when the gamble is unsuccessful. The designer tries to play a sure thing at the price of introducing additional hardware. It is important to remember that this implementation of the idea of achieving speed in computing by doing things concurrently is connected with local predictability in a computation. The state of the control and processor portions of the computer at any instant of time is highly dependent on the immediate past; and the expectation is that this dependency is useful in expediting what is desired to be done in the immediate future. This point must be borne in mind when we consider using such a computer as a central processor for a time-sharing system.

Not only has computer hardware changed in the last decade but so has computer software. Many computer languages have been devised along with assemblers and compilers. It is not clear that progress in software has kept up with progress in hardware. In particular it seems difficult to create a compiler capable of producing a program which exploits the concurrency of an advanced computer as well as can be done by a programmer. On the other hand, many executive systems do manage to keep input-output going efficiently and concurrently with the use of the central processor on other problems. These developments have had a number of aims, including the aim of making the computer more accessible to the user. This aim has not been fully realized because in practice the user is forced to deal with the computer via a monitor system—a practice dictated by efficiency of operation considerations. The monitor, however, introduces barriers between the problem poser and the computer which in many cases do not allow the machine to be fully or easily exploited in dealing with a particular problem.

2. Reasons for Considering Time-Sharing Systems

The need for removing barriers between the problem poser and the computer—or, more positively stated, the need for providing better man-machine interaction—is one of the major reasons for considering time-sharing systems. In order to see why this need exists, we must review some of the fact and fiction concerning the use of computers in dealing with large-scale scientific computations.

Such a use of a computing machine has been characterized by saying that "there is a relatively large amount of arithmetic done on relatively small amounts of data, and the output volume is also relatively small." Let us look in some detail at what is involved in finding the numerical

"solution" of an $m(= 1, 2, \text{or } 3)$-dimensional time-dependent problem in hydrodynamics when Lagrangian coordinates are used. There are $m + 1$ independent variables and $2m + 1$ dependent variables, since the Eulerian coordinates of the particle paths must be computed, the velocity field must be determined, and one thermodynamic variable must be calculated in addition to the density. The density is of course known if the particle paths are known. Each dependent variable must be determined as a function of the time.

It is not unreasonable to require that the extent of each spatial variable be divided into between 10 and 100 mesh points. The amount of data D which has to be stored in the computer at a given time (not necessarily in the high-speed memory) is then

$$30 \leq D \leq 7 \cdot 10^6 \tag{1}$$

words, where the lower limit holds for a one-dimensional problem with a 10-point spatial mesh and the upper limit holds for a three-dimensional problem with a cubical mesh having 100 points to the side.

The amount of calculation involved in determining the values of all the dependent variables at a mesh point in the time step from t to $t + \Delta t$ depends on the number of dimensions. We may assume that 75 arithmetic operations take place in computing all the dependent variables for a one-dimensional problem, 125 for a two-dimensional one, and 175 for a three-dimensional one. Hence n, the total number of operations per time step, will be

$$7.5 \times 10^2 \leq n \leq 1.75 \times 10^8.$$

The number of time steps taken in a given problem is usually a multiple K of the number of mesh points on a side. Hence for the whole problem the total number of operations is KN, where

$$7.5 \times 10^3 \leq N \leq 1.75 \times 10^{10} \tag{2}$$

and K may be as large as 30. (The quantity K is the number of times a sound wave will traverse one dimension of the fluid under consideration.)

Now, if a calculation similar to the one described above is done by inputting the amount of data D and performing KN operations, where D and N satisfy the inequalities (1) and (2), and then outputting an amount of data less than or equal to D, then it is indeed true that "there is a relatively large amount of arithmetic done on relatively small amounts of data, and the output volume is also relatively small." Note that the amount of computation per input data word varies from 250K for a one-dimensional problem to 2,500K for a three-dimensional one.

The method described above of inputting, running, and outputting does obtain when the physics of the problem is understood, the mathe-

matical method of solving is stable, and the code is debugged. Even in such cases the run time of the problem may be so long that it must be interrupted before completion, and thus there may have to be a number of input and output stages instead of one of each. Further, when the problem poser desires to "interact" with the calculation—that is, desires to modify some parameter (e.g., the time step) in the problem or to change the course of the computation in some other way (a way he may not be able to foresee when he is writing his program)—he may want to do this on the basis of results achieved to date, and this will require output and further input.

However, computations in hydrodynamics are not all done on problems in which the physics is understood; nor are foolproof methods known or foolproof codes written without a trying period of debugging. When the user is working toward the ideal state described above, the input-output demands are very much increased and may rise to the level of one input and one output for a relatively small number of time steps.

When one operates a computation facility in a batch-processing mode in which the user is provided the output on a printed page which he must use in some manner before he interacts again with the computer, very serious problems arise. The fact that at least one installation acquires its paper for such use in freight car lots (and measures the output in the height in feet of the stack of folded paper provided to the user) gives an indication of the seriousness of the problem and raises questions as to how efficient is batch processing and what is a true measure of efficiency.

Partly because a part of the input-output function can be done off-line, there is a tendency to rate the capacity of a computing system by the speed and memory capacity of the central processor or processors involved. It is certainly true that these quantities play a role of overriding importance in the "production" phase of a computation. Many problems now being done as a matter of routine could not be considered for numerical solution until computers were provided with sufficiently fast arithmetic units. We are, however, faced with the following situation: At present, in order to interact with such a computer, either in getting ready for production or even during the course of a production run, a large and perhaps unacceptable amount of time must elapse because we are using batch-processing methods to keep the central processor working.

It is important to decrease the time it now takes a scientific problem poser to interact with a computer when he is in the debugging phase of his work. This need may also exist in the production phase because the speed of present-day computers is such that a very large problem can now be done in the time that it previously took to do a moderate-sized one. I submit that we are not yet accustomed to properly formulating very large problems—or any large-scale endeavors—and must depend

on a certain amount of intervention in the course of the computation. In evidence of this I quote my own feelings, that I am convinced are shared by many people, on the completion of a large-scale computation. These feelings are summarized by the statement: "I now know how that computation should have been organized." Usually, the situation is left at that, and nothing is done about reorganizing the computation. If there had been more possibility of interacting with the computer even in the production phase, these afterthoughts would not have to be afterthoughts.

Time-sharing offers the promise of providing the scientific problem solver with faster interaction times—that is, with better man-machine interaction capabilities—and proposes to keep the computer busy during the time he is cogitating about the nature of the interaction. This double offer is therefore of great interest to the problem solver and to the manager of the computation facility.

Many facilities are faced with another serious problem which time-sharing offers a promise of alleviating. This problem is the following one: Some users of the facility have problems which demand computers of the greatest speed and capacity. However, such users do not provide enough work to saturate the computer all day, every day. The nature of the computing equipment acquired is determined by the nature of these problems which constitute the peak-load on the facility. The smaller problems can be, and of course are, done on the computer. If they are done in a manner where each user has sole access to the computer, the capacity of the computer is not fully utilized, and the user is faced with long turn-around times. It is proposed that under time-sharing the capacity of the computer be distributed to a number of smaller users whose total demand is commensurate with the capacity of the computer, who will be dealt with almost simultaneously, and who will be provided with response times of the order of seconds instead of hours.

The problem of providing shorter response times to solvers of smaller problems can of course be met by providing groups of these with separate computers of smaller capacity. It is an article of faith that needs to be examined in great detail—that one large computer is cheaper to build, maintain, and operate than a number of smaller ones. Even if one argues that, when the cost of the peripheral equipment and operating system overhead is taken into account, the economics of the situation are not in favor of the single large installation versus a number of small ones, one is left with two major advantages of the large system. One of these is the added ability to handle large problems of the type discussed above. The other is economical provision of a large library of programs and even data. If separate computer installations are going to share software and data by using modern communication networks, then the cost of each installation will rise, and the assumed economic unbalance will be redressed.

3. Design Problems for Time-Sharing Computers

If one accepts as one reason for the existence of a time-sharing system the provision of better man-machine interaction capability, then one is immediately led to the following questions: How shall the information to which the problem solver is to react be presented? How shall the problem solver communicate with the machine, that is, what language should he use and what communication channels should be made available to him?

In answering these questions, we may distinguish between the two phases of problem solving mentioned earlier—production and debugging. The distinction is not an absolute one, and remarks made concerning one phase may apply to the other.

Let us return to the hydrodynamics problem and suppose that the code is generating data purporting to describe the state of the fluid as a function of the time. The problem solver may wish to assure himself that the numerical method he is using is stable for the time-step size he has prescribed. In many non-linear partial differential equation problems this assurance cannot be provided by a prior mathematical analysis. Recent results enable us to state criteria for the linear variable coefficient case, but the non-linear case is still beyond analysis. As a result, the numerical values of the dependent variables must be monitored and watched for oscillations or other suspicious behavior. Suspicious behavior may be hard to describe, especially if the problem involved may have a physical instability which is being studied.

In any case there is a need for examining thousands of numbers, and it is clear that the reaction time cannot be small, if even before these numbers can be examined they all have to be printed.

The obvious thing is for the problem solver to have the machine do those mechanical things that he would do with the data—plot it, scan it to determine the "interesting" things about the data, and then present those portions to him in some quickly absorbable form. The difficulty here is to define "interesting," for if this could be defined in an *a priori* fashion, there would be no need to interact with the computer.

The existence of graphical devices and lightpens suggests one method of interacting with a computer in scientific problem solving: that the computer be asked to graph a portion of the data, and the problem solver be given the ability to circle a portion of the graph which he wishes to view in more detail. After the portion is circled, the computer presents him with the expanded graph he desires. He should of course be provided with the ability to set the scales used in graphing. The graphical techniques for talking to the computer should supplement other methods of making requests to the computer. We should have a graphing device responsive

to a specific request such as "plot the contents of memory positions $x = n, \cdots, n + k$ as a function of x" or "plot the array called X as a function of the one called Y." We would also like it to be responsive to a vague request such as "plot the array $X(n)$ as a function of n for those values of n where X is changing rapidly."

The graphical display problem is an old one, and many ingenious displays have been developed. The software problems associated with them are also in an advanced state of solution. There are, however, some novel software and hardware problems involved in incorporating these developments in a time-sharing scheme—especially one in which the displays are located at a site remote from the central computer. In such a case we are faced with a data transmission problem and the provision of hardware to buffer the data and generate the display. If transmission lines with low speeds of transmission are to be used, the amount of data sent from the central computer should be relatively small; yet one may be interested in displaying widely varying quantities. An elaborate computer is not needed for the graphical display generation. A very large memory which can communicate to the graphical display and to the central computer is of course needed.

The debugging phase of solving a hydrodynamics problem is usually considered to involve relatively small amounts of data or programs which can be communicated to and from the computer by use of a typing device. If this device is located at a remote site and if low transmission rates are to be used, then the central facility must have a large back-up memory. This is so because, even though relatively little information is to be transmitted, if this is to be meaningful, it must be part of a larger amount of information. The latter must be available to the central computer. It is already available to the problem solver, for he presumably has a record of the previous communications to the central computer and has some idea of what he is trying to do.

The back-up memory must have the capacity to store all the information pertinent to all the users sharing the computer and be able to make the information for a particular job available for use within prescribed time limits. Indeed, the number of users that can use a time-sharing facility is limited by the capacity of this memory and by the time it takes to make a portion of its contents available to the main memory of the system.

If the problem solver is going to interact with the computer, and if the machine is going to be shared by other users, it is evident that the language used by the problem solver must be quite different from the machine language. In particular, physical memory addresses of data and instructions will not be known to him. Since he is interested in the behavior of certain variables, he would like to refer to these by name and not be burdened with anything else. This suggests that the software and hardware

design allow this and be designed so that memory relocation and protection are done for him.

The hardware and software must have another characteristic: that they be as capable of dealing with non-numeric data as they are with numeric data. For, in the interaction phase, a great portion of the central facility's effort will be devoted to translating the general statements made by the problem solver into strings of machine instructions.

It must however be remembered that the problem solver must be concerned with what is going on in the arithmetic unit. If he is anxious about the stability of a numerical procedure, he may need to know the size of both the most significant and the least significant portion of the product before it is rounded and stored. Thus the language provided to the user must include the capability of referring to the inner parts of the computer as well as the capability of referring to "arrays" or "variables."

A large part of the nostalgia some computer users have for the good old days of the von Neumann machine without monitors is due to the fact that language creators and system programmers have not provided users with the ability to query sub-units of computers. This type of query is essential when one is trying to disentangle mathematical method stability from phenomenological stability; and unless it can be decided which instability is being manifested, the whole computation may be worthless.

3
Computer Utility: Objectives and Problems*

R. M. FANO
Massachusetts Institute of Technology

[ABSTRACT]

The original motivation for the development of time-sharing systems was to make it possible for several people to access simultaneously a powerful computer in a manner that would give each of them the illusion of having a private computer at his disposal. Actual experience with the operation and use of time-sharing systems has brought to light other virtues of such systems that will be in the long run much more significant, namely, their use as "thinking tools" and their role as powerful means of intellectual communication between people. These virtues are responsible for the growing interest in computer utilities. The concept of a computer utility was discussed from the point of view of the user, the manager, the designer, and society as a whole.

---

* The manuscript of this paper was not available at the time this book was published. A paper by Professor Fano on the same general topic appeared in the *IEEE Spectrum*, January 1965, pages 56–64. Work reported therein was supported by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number Nonr-4102(01).

SESSION II

Man-Computer Interface

4

A User's View of a Functionally Oriented On-Line System*

G. J. CULLER
University of California, Santa Barbara

[ABSTRACT]

With systems of the type considered here, a user has control directly over the mathematical organization of his problem. There is presently operating at TRW Systems a four station on-line system which is functionally oriented. It is highly interactive in the sense that each user gets service upon immediate depression of a console key. This system is an outgrowth of earlier systems used experimentally at Bunker-Ramo in Canoga Park. At present the University of California at Santa Barbara has a similar system containing further developments operating with a classroom of sixteen (16) consoles and, in addition, remote users at Harvard University and at the University of California at Los Angeles. The remote users enter the system through the use of Western Union data sets. The style and character of the system have been developed in response to a need expressed by a very in-group of users (mostly physicists) with mathematical applications.

* The manuscript of this paper was not available at the time this book was published.

5

JOSS: Experience with an Experimental Computing Service for Users at Remote Typewriter Consoles*

J. C. SHAW
The RAND Corporation

INTRODUCTION

In discussions of on-line versus off-line use of computers, attention frequently focuses on relative costs to the neglect of relative benefits. An experimental computing service, at remote typewriter consoles, was made available to the staff of The RAND Corporation to explore the value of on-line use of a computer. JOSS** (Johnniac Open-Shop System) monitors ten typewriters and serves up to eight users concurrently. It has been in daily use since January 1964. (An earlier version saw limited use beginning in May 1963.)

Since 1963, and even earlier, other on-line, time-shared, general-purpose computing systems have come to fruition. The largest of these, at the Massachusetts Institute of Technology [1] and at the System Development Corporation [2], provide computing services for a broad spectrum of computing requirements.

In contrast, JOSS is limited by the capacities of the Johnniac, a Princeton-class computer constructed in 1950–53. T. O. Ellis and M. R. Davis painstakingly designed a communication system to connect familiar typewriters; and the author worked hard to design a smooth language for specifying small numerical computations. We believe the resulting improved access—not raw computing power—leads users to prefer JOSS to alternative conventional computing services for many problems.

** JOSS is the trademark and service mark of The RAND Corporation for its computer program and services using that program.

Descriptions of the Johnniac, the typewriter communication system, and the language appear elsewhere [3–7]. Here, we deal primarily with the philosophy of the system and our users' experiences with it.

HARDWARE

The existence of an in-house computer nearing the end of its useful life provided the opportunity to experiment. Indeed, the Johnniac was used experimentally as soon as commercial computers became better able to meet RAND's computing needs. The 4,096-word magnetic-core memory, 12,288-word magnetic drum, and an austere order code presented a stiff challenge to produce any workable on-line time-shared system. It was necessary to construct a communication system to monitor and control remote typewriters and to buffer data transmissions. (In modern equipment, of course, this communication requirement may, in some cases, be better met by interrupting the central processor than by using a communications processor.)

The typewriter chosen was the IBM Model 868 with a paging control. (Teletypewriters were judged unsatisfactory, and the IBM SELECTRIC® appeared too late for consideration.) Cooperative maintenance has overcome the initial troubles with the typewriters. The character set was carefully chosen to preserve all the standard punctuation marks. However, brackets and the symbols for arithmetic operations and numerical relations replaced the less frequently used special characters. A two-color ribbon switches automatically to black for output and to green for input. These features facilitate typing and reading the JOSS language and contribute to the philosophy that the user shall interact with JOSS as with a computing aide, using a natural, high-level language at all times.

SOFTWARE

To carry out the philosophy of presenting JOSS to the user as a computing aide and the *only* active agent with which he communicates, it was necessary to "hide" the Johnniac from the user and to present instead the image of a person interpreting instructions and remaining in control of the situation, no matter how senseless those instructions may be. This contrasts with a code-checking session in which one is concerned with machine-language instructions and representations of objects at the bit level. There, one is reminded of nothing so much as an experimenter carefully setting up an elaborate electronic apparatus, then throwing the switch for a smoke test—because the machine-language interpreter is prepared to manipulate only pieces of complex representations. The programmer who tries to repair the damage of a first test in preparation for a second is really doing it himself, because the available interpreter operates at

too low a level for him to get any feeling of linguistically directing an agent. This effect is so strong that in programmers' jargon *routines* are personified as active agents, seldom the machine-language interpreter.

Not so with JOSS. The programs the user gives JOSS are strictly static, passive specifications of computing and typing processes that JOSS carries out (Figure 1). If the user specifies a process imperfectly, JOSS will call it to his attention with an error message and stand by to help with the changes. Frequently the user can direct JOSS to make the correction and continue without having to start over at the beginning of the program. Sometimes the user directs JOSS to carry out a sensible process but not the one intended. Then he himself must detect this from JOSS's actions and results.

JOSS lacks the problem-solving capacity to carry on a sophisticated conversation. For its comments, it simply selects from a stock of forty "canned" messages. But the timeliness and appropriateness of its remarks

```
*   PRIME NUMBERS

1.1 Do part 2 for d = 2(1)ip[sqrt(x)] if x > 3.
1.2 Type x if x ≠ 0.

2.1 Set x = 0 if fp(x/d) = 0.

Do part 1 for x = 2(1)50.
       x =       2
       x =       3
       x =       5
       x =       7
       x =      11
       x =      13
       x =      17
       x =      19
       x =      23
       x =      29
       x =      31
       x =      37
       x =      41
       x =      43
       x =      47

Type all steps.

1.1 Do part 2 for d = 2(1)ip[sqrt(x)] if x > 3.
1.2 Type x if x ≠ 0.

2.1 Set x = 0 if fp(x/d) = 0.

Delete all.
```

FIGURE 1

```
1.1 Set x(1) = [-b+sqrt(b*2-4•a•c)] / [2•a].
1.2 Set x(2) = [-b-sqrt(b*2-4•a•c)] / [2•a].
1.3 Type x(1), x(2) in form 1.
Set b = -1.
Set c = -6.
Do part1.1.
Eh?
Do part 1.1.
Error above:  Part number must be integer and 0<part<10*9.
Do part 1.
Error at step 1.1:  a is undefined.
Set a = 0.
Go.
Error at step 1.1:  I have a zero divisor.
Set a = -1.
Go.
Error at step 1.1:  I have a negative argument for sqrt.
Set a = 1.
Go.
Error at step 1.3:  I can't find the required form.
Form 1:
a =___    b =___    c =___    roots:    .____    .____
Go.
Error at step 1.3:  I have too few values for the form.
1.3 Type a, b, c, x(1), x(2) in form 1.
Go.
Error at step 1.3:  I can't express value in your form.
Form 1:
a =___    b =___    c =___    roots: ___._____.____
Go.
Error at step 1.3:  Your fields run together in the form.
Form 1:
a =___    b =___    c =___    roots: ___.____ ___.____
Go.
a =  1    b = -1    c = -6    roots:   3.0000   -2.0000
Do part 1 for c = -1(1)-4.
Error above:  Illegal set of values for iteration.
Do part 1 for c = -1(-1)-4.
a =  1    b = -1    c = -1    roots:   1.6180    -.6180
a =  1    b = -1    c = -2    roots:   2.0000   -1.0000
a =  1    b = -1    c = -3    roots:   2.3028   -1.3028
a =  1    b = -1    c = -4    roots:   2.5616   -1.5616
2.1 Do step 2.1 for i = 1(1)100.
Do step 2.1.
I need more storage space.
Delete step 2.1.
Go.
It's a mess.  Let's start over.
Go.
Error above:  You haven't told me to do anything yet.
Do part 1 for c = -5.
a =  1    b = -1    c = -5    roots:   2.7913   -1.7913
Please wind up your work and turn off as soon as possible.
Delete all.
```

FIGURE 2

give a feeling of interacting with a person. After a year's experience with neutral, impersonal messages, we introduced personal pronouns and some informal diction (Figure 2). This was done not facetiously but rather to reinforce the model. No objections to the more personal messages have been voiced.

JOSS not only permits but requires high-level interaction between man and machine. The user has no way of referring to the internal representation of any of the objects JOSS deals with: value assignments, steps, or forms. The details of the representation are completely sealed off. Thus, except for hardware malfunctions, no JOSS "experts" are required to explain the behavior of the system in terms of machine-level operations. Inside knowledge is of no advantage to the user.

JOSS hides the binary nature of the Johnniac by performing arithmetic such that the user sees floating-decimal operations with nine digits. No conversion errors are introduced on input or output, and internal arithmetic is familiar. The user may ask JOSS to add .1 ten times, and the result is indeed 1. Exact arithmetic is performed so long as results do not exceed nine decimal digits. Longer results are rounded in the conventional way.

The user directs JOSS in three basic activities: numerical processing, editing, and typing. The language contains elements to facilitate all three. The specification of forms for typing out results is done by presenting JOSS with a pattern for the output line. This is only one component of the many that are used in sophisticated report generators. The remarkable thing about these forms is the general adequacy of just two types of field specification: one for fixed-point output and one for scientific notation. If the user fails to specify a form for the output, JOSS chooses a reasonable one: JOSS types in fixed-point form if the magnitude of the result is between .001 and 1,000,000; otherwise, JOSS types in scientific notation.

TRAINING

The use of the JOSS system is best taught by example. Though several documents and a motion picture show the uses of JOSS in standard types of numerical work (Figure 3 serves as a check list of what the system can do), an hour's demonstration at a typewriter console is still the most effective introduction. With such a demonstration and access to the documents, a new user enters a phase in which he can use the system productively for solving simple problems. Meanwhile he familiarizes himself with the full power of the system by informal exploring.

The user receives no formal description of the language, since such a description would be at least as difficult to learn as the JOSS language itself. A JOSS representative (not a computer expert) is available in each department to assist new users.

DIRECT or INDIRECT

```
Set x=a.

Do step 1.1.
Do step 1.1 for x = a, b, c(d)e.
Do part 1.
Do part 1 for x = a(b)c(d)e, f, g.

Type a,b,c,_.
Type a,b in form 2.
Type "ABCDE".
Type step 1.1.
Type part 1.
Type form 2.
Type all steps.
Type all parts.
Type all forms.
Type all values.
Type all.
Type size.
Type time.
Type users.

Delete x,y.
Delete all values.

Line.

Page.
```

INDIRECT (only):

```
1.1 To step 3.1.
1.1 To part 3.

1.1 Done.

1.1 Stop.

1.1 Demand x.
```

DIRECT (only):

```
Cancel.

Delete step 1.1.
Delete part 1.
Delete form 2.
Delete all steps.
Delete all parts.
Delete all forms.
Delete all.

Go.

Form 2:
dist. = .....  accel. = __.___

x=a
```

RELATIONS:

= ≠ ≤ ≥ < >

OPERATIONS:

+ - · / * () [] ||

CONDITIONS:

if a<b<c and d=e or f≠g

FUNCTIONS:

```
sqrt(a)    (square root)
log(a)     (natural logarithm)
exp(a)
sin(a)
cos(a)
arg(a,b)   (argument of point [a,b])
ip(a)      (integer part)
fp(a)      (fraction part)
dp(a)      (digit part)
xp(a)      (exponent part)
sgn(a)     (sign)
max(a,b)
min(a,b,c)
```

PUNCTUATION and SPECIAL CHARACTERS:

```
. , ; : ' " _ # $ ?
___.____ indicates a field for a number in a form.
........ indicates scientific notation in a form.
# is the strike-out symbol.
$ carries the value of the current line number.
* at the beginning or end kills an instruction line.
Brackets may be used above in place of parentheses.
Indexed letters (e.g. v(a), w[a,b]) may be used above in place of x, y.
Arbitrary expressions ( e.g. 3·[sin(2·p+3)-q]+r ) may be used above
  in place of a, b, c, ... .
```

FIGURE 3

Most of the nearly 200 JOSS users had no prior direct programming experience. Those who had experience betrayed this fact by the style in which they first used JOSS. They frequently worked on a program till they felt that all eventualities were covered before telling JOSS to do *any* processing.

The novice has no conflicts with other programming systems and so has an easier time. Even grade-school children have shown an ability and eagerness to use JOSS on problems they understand.

The JOSS language *is* easy to learn. During the development of JOSS, its use was restricted to a select group of cooperative users who were asked to test the system during a period of rapid change. Near the end of that period, an unauthorized senior staff member was observed using JOSS before any training program had been set up. It turned out that he was taught by another unauthorized senior staff member who was taught by still another unauthorized senior staff member who learned by looking over the shoulder of an authorized user.

EXPERIENCE

JOSS users face three serious limitations in the experimental system: frequent malfunctions in the Johnniac; an irregular schedule; and no provision for filing programs for use in a later session. Speed limits the user only occasionally.

The Johnniac performs perfectly about 95 per cent of the time scheduled for JOSS service. This is perhaps better than one should expect of an aging vacuum-tube computer. In conventional service it is adequate, but in on-line work the system usually "crashes" every other day. The loss is limited by the size constraints on programs (about three typewritten pages), but any disruption is irritating.

JOSS is incompatible with other uses of the Johnniac. No background processing occurs to absorb otherwise idle time. At the beginning of the experiment, we expected to dedicate the computer to JOSS service. But other experimental work led to important applications that could not be "bumped." So, the regular schedule for JOSS is only from 10 a.m. to 6 p.m., Monday through Friday.

The Johnniac has no magnetic tape or disk file for program storage. We rejected paper tape punches and readers at the remote stations as expensive and inconsistent with the desired central files. We also rejected a punched card file because it would require an operator, and punching and reading would interfere with the basic service.

These shortcomings in reliability, schedule, and filing should be kept in mind while interpreting the following statistics for a week that was free of malfunctions:

Statistics for April 1–7, 1965, 10:00 a.m. to 6:00 p.m. (40 hrs.)

1. JOSS worked for 77 different users in the week.
2. On the average day, 31 users conducted 49 sessions.
3. The sessions averaged 58 minutes, with 50 per cent less than 35 minutes.

4. The system was saturated with 8 users 18 per cent of the time, and 4 per cent of the time someone was waiting in the queue.
5. There were 5.7 simultaneous users, on the average.
6. The input rate was 1.9 lines per minute per user. The output rate was 3.9 lines per minute per user. That is, there was a line of input or output every 10 seconds for every user on average.
7. JOSS worked on user blocks in core memory 57 per cent of the time. (This tends to 75 per cent during busy periods.)

The statistics are objective though indirect evidence of the usefulness of JOSS. Direct but subjective evidence is found in the comments of the users. One said he was "ready to give up the slide rule." Another mathematician claimed to have used the service for "everything from finding the product of two real numbers to solving fifteen simultaneous transcendental equations" but then told how, on one problem, the easy access to computing power led him to postpone a more thoughtful approach that finally succeeded. A senior electronics engineer, arguing for a replacement of the Johnniac, wrote:

> JOSS is becoming essential to our output like paper, or coffee. It speeds up calculations; makes it easy to try experiments. It is the greatest, when it works. I have heard it compared favorably to beer and referred to as RAND's most important fringe benefit. People adjust their lives to fit around JOSS . . . . No use coming to RAND before 10:00 a.m. when JOSS arrives, in fact noon or after 5:00 p.m. is a better time, JOSS is less busy. When JOSS starts typing answers, the titillating pleasure is equalled only by the ensuing anguish when JOSS breaks off into gibberish or goes away commending your program to oblivion. We can hardly live with JOSS, but we can't live without it. We're hooked.

CONCLUSION

Sixteen months should be long enough for the novelty of JOSS service to have worn off. Most RAND staff members having at least occasional computing problems have been exposed to the system. The statistics developed from the minute-by-minute log of activity show that many users find the investment of their time and effort in the on-line direction of JOSS to be rewarding.

At this point, several challenges arise:

To take the system out of the experimental stage and into a regular economical service on modern reliable equipment;

To extend input-output capabilities to devices other than typewriters;

To extend the range of application beyond small numerical problems while preserving the best features of JOSS.

ACKNOWLEDGMENTS

The JOSS experiment involved many individuals and could not have attained its modest goals without such contributions as: the stimulus from A. Newell; the authorization by P. Armer and W. H. Ware; the communications development under T. O. Ellis and M. R. Davis; the project management of K. W. Uncapher; the maintenance efforts of R. I. Yoshimura and crew; the programming assistance of Leola Cutler and Mary Lind; the instruction of users by C. L. Baker, G. E. Bryan, and F. J. Gruenberger; and, certainly not least, the enthusiasm of the first users—O. A. Gross, N. Z. Shapiro, W. L. Sibley, A. C. Smith, and J. D. Williams.

REFERENCES

[1] CORBATO, F. J., *et al.*, *The Compatible Time-Sharing System: A Programmer's Guide*, Massachusetts Institute of Technology Press, Cambridge, Massachusetts, 1963.

[2] COFFMAN, E. G., JR., J. I. SCHWARTZ, and C. WEISSMAN, "A General-Purpose Time-Sharing System," *AFIPS Conference Proceedings* (1964 SJCC), Vol. 25, Spartan Books, Baltimore, Maryland, 1964, pp. 397–411.

[3] BAKER, C. L., *JOSS: Scenario of a Filmed Report*, The RAND Corporation, RM-4162-PR, June 1964.

[4] "The JOSS System: Time-Sharing at RAND," *Datamation*, Vol. 10, No. 11, November 1964, pp. 32–36. [This article is based on Ref. 3.]

[5] SHAW, J. C., "JOSS: A Designer's View of an Experimental On-Line Computing System," *AFIPS Conference Proceedings* (1964 FJCC), Vol. 26, Spartan Books, Baltimore, Maryland, 1964, pp. 455–464; also, The RAND Corporation, P-2922, August 1964.

[6] SHAW, J. C., *JOSS: Examples of the Use of an Experimental On-Line Computing Service*, The RAND Corporation, P-3131, April 1965.

[7] SHAW, J. C., *JOSS: Conversations with the Johnniac Open-Shop System*, The RAND Corporation, P-3146, May 1965.

DISCUSSION

J. C. MCPHERSON: What, in the light of experience, are the most important changes you would like to make in a follow-on system?

J. C. SHAW: I would like to get out, myself. Fortunately, this is a very clean breaking point. Mr. C. L. Baker heads a team with the responsibility for JOSS-II on a PDP-6 which has been ordered to replace the Johnniac. One of the obvious additions to the language is the summation operator, which is very powerful in compressing programs.

Function definition is another. It is quite easy to define procedures for JOSS, but the ability to define a function and use the function name in other expressions would be a useful extension. These can come easily with modern equipment and greater storage capacity, but it is a challenge to preserve the smoothness of the system while extending the language.

J. M. BENNETT: Is it proposed to be able to pass the programs across to the normal computing queue?

J. C. SHAW: The PDP-6 will be dedicated to JOSS service. There will ultimately be ways, other than via the typewriter, of getting data out, but, at the moment, there are no plans for taking a JOSS program out and, say, compiling it for another system.

D. FARBER: In JOSS-II you are going to have filing of programs. Have you thought how to do that without needlessly complicating the language or the operation of the language?

J. C. SHAW: Filing will be very simple-minded: just ask JOSS to file the program, get a name for it, and call for it by name at another session.

QUESTION: In this system, do you give any priority to any particular station or particular users in considering how to get it on-line as soon as possible?

J. C. SHAW: No, they are all treated equally—first come, first serve.

H. KRENN: How much core space is required for your system?

J. C. SHAW: Johnniac has a 4,096-word core, and the system is 6,000 words. Only half of the system resides in core storage. JOSS goes to the drum to get infrequently used routines such as output routines.

R. R. FENICHEL: I wonder how reliable your statistics are. I don't mean to criticize you in this regard, because statistics gathering for time-sharing systems is certainly a very difficult thing to do. But in the case of JOSS, I think the problem may be more severe than usual.

The several users of JOSS are not each permanently associated with a priority, a set of privileges for using various parts of the system, a dollars-and-cents time account, and a set of personal program and data files. Thus, there is no incentive, as I see it, for one JOSS user ever to announce to the system that he is taking a console over from another user.

I suppose that the real basis of my question is this: I was out at RAND during the week when you were gathering statistics.

J. C. SHAW: Didn't you use your own initials?

R. R. FENICHEL: No, I didn't. I used JOSS for about an hour, and I never signed on. Whatever else should be done with your statistics, the "77" should be upped to "78."

J. C. SHAW: It is true, perhaps, that the statistics on users are cluttered a little by demonstrations, but I think that is in the noise level.

E. E. DAVID, JR: I guess one of the questions Mr. Fenichel had in mind was why two or three or ten users can't use the system under one log-in. Actually the number you gave might be conservative.

J. C. SHAW: Yes, they can. But we feel vanity will impel them to use their own initials.

6
Linguistic Problems of Man-Computer Interaction*

ANTHONY G. OETTINGER
Harvard University

It's a great pleasure to follow Messrs. Fano, Culler, and Shaw in this symposium. The time sequence has symbolic significance for me, since I choose to interpret it as paralleling my standing on their shoulders in the research on man-computer interaction now going on in my laboratory. We are at this moment using Project MAC console and display facilities to embody as much as we can of the spirit of the on-line computer already realized in several forms by Culler. Our implementation, however, incorporates what would, I think, appear to Culler as certain heresies, for instance, in the direction of JOSS.

Before concentrating on differences which reveal certain of the key problems in man-computer communication, let me first comment on the common enthusiasm linking us with these people and many others, such as Roberts, Ross, and Sutherland, in the common enterprise of exploring the processing by men and computers of mixtures of graphic, symbolic, and conventional linguistic messages. The words I shall use are not my own, since I recently discovered that Marshall McLuhan, far from being the alien Madison Avenue type I took him for, when I ran across reviews of his books *The Gutenberg Revolution* and *Understanding Media,* is indeed a kindred spirit and perhaps the strongest exponent of sensible views on automation outside the sensible portion of the computer profession.

With an eye particularly to television, but explicitly with other implications in mind, here is what McLuhan says:

---

* The preparation of this paper was supported in part by Advanced Research Projects Agency under Contract SD-265 and by National Science Foundation under Grant GN-329.

> The term "literature," presupposing the use of letters, assumes that verbal works of imagination are transmitted by means of writing and reading. The expression "oral literature" is obviously a contradiction in terms. Yet we live at a time when literacy itself has become so diluted that it can scarcely be invoked as an esthetic criterion. The Word as spoken or sung, together with a visual image of the speaker or singer, has meanwhile been regaining its hold through electrical engineering. A culture based upon the printed book, which has prevailed from the Renaissance until lately, has bequeathed to us—along with its immeasurable riches—snobberies which ought to be cast aside. We ought to take a fresh look at tradition, considered not as the inert acceptance of a fossilized corpus of themes and conventions, but as an organic habit of re-creating what has been received and is handed on.

Although we may not yet have formed and may never form that "organic habit of re-creating what has been received and is handed on," the need to do so fortunately is overtly thrust upon us.

I alluded earlier to our Shavian deviationism from the Culler line, a matter concerned with what is perhaps one of the strongest of the dynamic oppositions that shape natural languages. With respect to these oppositions, natural languages seem to have achieved a kind of dynamic stability, which, in the past ten years, has been described primarily by Mandelbrot. The means of reaching stability are still obscure to linguists and psychologists alike. Zipf attempted, 30 years ago, to explain them by his quasi-mystical Principle of Least Effort. With few exceptions, notably Jakobson, serious (read "stuffy") linguists continue to ignore the problem of the dynamics of change and stability in language.

Quine has given eloquent and concise expression of one fundamental aspect of the problem:

> In logical and mathematical systems either of two mutually antagonistic types of economy may be striven for, and each has its peculiar practical utility. On the one hand we may seek economy of practical expression—ease and brevity in the statement of multifarious relations. This sort of economy calls usually for distinctive concise notations for a wealth of concepts. Second, however, and oppositely, we may seek economy in grammar and vocabulary; we may try to find a minimum of basic concepts such that, once a distinctive notation has been appropriated to each of them, it becomes possible to express any desired further concept by mere combination and iteration of our basic notations.

The most clear-cut and quantitative description of the range of choices available in resolving the antagonism Quine describes is given to us in Shannon's *Mathematical Theory of Communication.* Given the need to transmit or store a message worth b bits of information, we can calculate precisely how many characters of a large alphabet or how many more characters of a smaller alphabet would be needed to encode the same message. If messages are selected from a population according to some probability distribution, a well-known algorithm of Huffman tells us how

to encode each message in a given alphabet so as to minimize the cost of transmission. Contemporary coding theory is an elaboration of these basic ideas to accommodate a variety of other side conditions including, for example, the desire to minimize the transmission error rate.

I think it is fair to say that Shaw leans toward the side of small vocabulary but long expressions, while Culler enlarges his basic vocabulary to obtain shorter expressions. Beyond this kind of statement, however, there is nothing in this realm approaching even the limited theoretical understanding that Shannon's theory has given us of the information transmission process.

How does one, for example, trade off the ease of punching a single button in Culler's system to evaluate the sine—not of a single number, but of a whole interval—as opposed to the need in JOSS, first, to type several characters to name the function (a process requiring one to remember the name of the function and to spell it correctly at the time of punching several keys) and, second, the need to write explicitly some kind of loop expression to turn what is basically a point operation into a function operation? How to compare the value of some mnemonic name invoked with several key strokes to call in a main subroutine in FORTRAN against the ease of pushing the one button on level 13 where the subroutine is attached when, however, we may find it hard to remember which button we attached it to when we constructed it? How does one define a syntax maximizing the advantages offered by prefix notation at one extreme and fully parenthesized infix notation at the other?

In any case, either system is relatively easy to learn, and either is far more foolproof than many that preceded it. Ease of learning complicates the matter of value judgment, since once a subject is accustomed to either system, the other will seem repulsive, or at least more difficult. Objective analysis of such questions may therefore take us far from the austere mathematical simplicity of the noiseless communication channel into murky and controversial realms of experimental psychology.

Yet even here lies an opportunity.

In *The Act of Creation,* Arthur Koestler asks the question: "What lesson, for instance, could one expect neurophysiology to derive from astronomy?" Koestler then answers himself, by relating how Bessel, the astronomer of Bessel-function fame, having read about the dismissal of an assistant of the Astronomer Royal at Greenwich for consistently making observations differing from those of the Astronomer Royal by a half second to a whole second, was moved to compare his own records over a ten-year period with those of several astronomers and thereby to prove that "there existed systematic and consistent differences between the speed with which each of them reacted to observed events; he also succeeded in establishing the characteristic reaction time—called 'the personal equa-

tion'—of several of his colleagues." Fifty years later, Helmholtz, who had become acquainted with the work of astronomers on "personal equations," determined "that the rate of conduction of impulses in nerves was of a definite, measurable order—and not, as had previously been assumed, practically instantaneous."

I suspect that our common concern with man-computer communication, especially with its increasing attention to the pictorial and a continuing profound concern with *processes* rather than with the static snapshots of classical mathematics, will be a fertile source of new psychological discovery for those alert and lucky enough to be discerning at the right time.

While at present we are merely rediscovering what the psychologists already know about the value of direct and immediate control and direct and immediate response, two factors which largely account for the ease of learning systems even with embarrassingly thorny conventions concerning parentheses, we have new opportunities for attacking these questions with apparatus and from a point of view on the whole alien to the psychological community.

I think also that the expanding use of MAC, JOSS, or Culler-like facilities will lend itself to the study of problem-solving behavior in a setting far more natural than the artificial experiments of the psychologists or the unobservable activities of problem solvers at their desks.

What saves both Culler and Shaw from paying the high price of excessive economy, either of alphabet or of expression length and complexity, is the provision in both systems of a facility that obviously has played a major role in stabilizing both natural and artificial languages but whose nature is adequately understood by neither philosophers, linguists, psychologists, nor computer scientists. This facility is the ability to define new entities in terms of old and the closely related and equally ill-defined and ill-understood process labeled in mathematics as "simplification."

The "list" or "program" operation in Culler's system is such a definitional capability. Concomitant with definition comes abbreviation. What was a long sequence of button pushes is, through the act of attaching it to a single button, both defined as a new operation and abbreviated to a single button push. So, in natural language, neologisms supplant elaborate circumlocutions and descriptive statements, once a concept or object has become fixed enough for it to become clear that frequent naming will be required. A similar role is played by the numbering facility in JOSS which enables what is essentially a subroutine to be identified by a part number that subsequently may be referred to in a statement such as "do part 3." JOSS lacks means for defining a function of one or more arguments invokable by name elsewhere, but the LISP language, for example, has an explicit and extremely useful "define" operation,

and, on the MAC/CTSS system, new processes defined in terms of any of the languages available on the system may, once accepted, be called for by any member of a growing community in highly abbreviated form as system commands.

The need for conscious or unconscious abbreviation is recognized also in the design specification of PL/I stating "that ... every attribute of a variable, every option, every specification was given a default interpretation, and this was chosen to be the one most likely to be required by the programmer who does not know that the alternatives exist." The spirit is most laudable, but Oh, what problems are raised by the phrase "the one most likely to be required by the programmer who does not know that the alternatives exist"!

There is a great need for putting order in the chaos of definitional facilities, whether in the shape of subroutines and call statements, macros, explicit definition statements, and so on. The odds are about even for linguistics to contribute to computer sciences in this matter, or vice versa.

Similar obscurities range in the realm of simplification and the related matter of evaluation. Everyone is familiar with examples such as one recently given by Hamming, who points out that 9th-grade students may be required to "simplify" $1/\sqrt{a} + 1/\sqrt{b}$ to $\sqrt{ab}\ (\sqrt{a} + \sqrt{b})/ab$, although what is meant here by "simplicity" is by no means clear. Most of those who have worked on automatic algebraic manipulators have stubbed their toes on this wall. The shift to man-computer interaction simplifies the problem in the sense that a well-designed system should permit the man to use his uncanny intuitive faculties for making some kind of simplification, but it has not yet deepened our understanding of the notion itself. In fact, it is tempting to speculate that by treating ordering on a scale of simplicity in a manner akin to the treatment of the notion of computability—by Turing, Church, and others—one might prove that *whatever* definition of ordering one seeks, such an ordering cannot be more than a pre-order; that is, in any ordering of simplicity there will always be at least two statements such that if one precedes the other, the second also precedes the first. Such a result would at least let one proceed with muddling through in special cases with a clear conscience that no general solution can be found.

The importance of symbol manipulation in man-computer interaction, as elsewhere, emphasizes once again the key importance of the distinction between use and mention, as exemplified by the explicit use of a QUOTE statement in LISP and by the LIST operation in Culler's system which distinguishes the mention of a sequence of operations from the use or execution of this sequence. The complexity of the whole problem is well illustrated by certain elegant solutions of certain aspects of it in the new version of formula ALGOL developed by Perlis and his associates at

Carnegie Tech. In this system, expressions in their natural state are mentioned or manipulated, but they are used only as arguments of an evaluation operator which, in a natural way that seems to synthesize several hitherto unrelated notions, may (1) collapse a formula into a single number in the usual way when all its members are themselves completely specified as numbers, or (2) expand it when the values of parameters or variables in the expression are specified not as numbers but as formulas, and, finally, (3) simplify it, for instance, by making a substantial portion of a formula vanish if it consists of zero-multiplying an arbitrarily complicated formula.

In any of these three cases, however, the formula is used but not mentioned in the sense that the original formula is untouched and the result of the evaluation is stored elsewhere unless the address for the result happens to be that of the formula itself—an interesting case of self-reference, whose consequences need careful clarification.

Beside the matter of choosing the size of an alphabet, vast realms of unanswered questions appear when one considers further dimensions of the problem of what is good notation. It has long been recognized that the development of an apt notation is, in many instances, half the battle, for good notations, so to speak, have a life of their own and may often lead rather than follow thought, or notations may inhibit it, as exhibited in the following humorous example of Henle.

Henle, in an analysis of the persistent complaint of mystics that language is inadequate to express their insights, poses in a very amusing way a serious problem of symbolic geometry. He postulates a primitive tribe in the process of developing a rudimentary conception of algebra. Where we use letters of the alphabet, they use little triangles and squares, for example, "△" for "*a*" and "□" for "*b*." Where we would distinguish between lower-case and capital letters, they would distinguish, for example, between "△" and "△." Instead of representing $a + b$ by "$a + b$," they superimpose the symbols for the two variables as follows: $\overset{+}{\boxed{\triangle}}$. Henle goes on:

> A curious result ensues, from the use of this symbolism. Whereas we can easily state the very simple rule of arithmetic that $a + b = b + a$, this is impossible in their symbolism. If one tries to write it, one comes out with $\overset{+}{\boxed{\triangle}} = \overset{+}{\boxed{\triangle}}$ and this of course is indistinguishable from the formulations of $a + b = a + b$. Hence, they are bound to regard it as an immediate consequence of the law of identity: △ = △, obtained by substituting "$\overset{+}{\boxed{\triangle}}$" for "△". This result is inevitable and inescapable. With the symbolism at hand there is no way of distinguishing between "$a + b$" and "$b + a$". At first one might be tempted to write one "$\overset{+}{\triangle}$" and the other "$\overset{+}{\boxed{\triangle}}$"; but this will not succeed. By the convention established, "△" and "△" are different symbols so that if one complex is equivalent to $a + b$, the other is equivalent to $b + c$. You may object that this is a very poor symbolism: I shall not dispute the point. All I claim is that it is a symbolism which might be used.

Henle then goes on:

> Not long ago, a young genius tried to treat subtraction generally, to work out laws of subtraction for all numbers. Using the notation "$\boxed{\triangle}$" for "$b - a$," he wished to enunciate the rule that in general $b - a \neq a - b$. He wrote that, in general, $\boxed{\triangle} \neq \boxed{\triangle}$. He was accused, of course, of violating the law of identity, the most sacred law of logic, and was promptly condemned to death. He met his end rather pensively, confusedly shaking his head. He admitted that what he wrote seemed to contradict itself, but somehow, he was sure, it didn't.

The matter of choice of notation is one where it is of prime importance to establish a "habit of re-creating what has been received and is handed on." The prevalent assumption that it is most natural, since men communicate with one another in a vernacular such as English, for them to communicate in English to a computer needs careful examination. First of all, it is not wholly true that men communicate with one another in English under all circumstances. One need only witness the helplessness of two chemists or mathematicians in conversation away from a blackboard to understand this. We rebel against the constraints that badly designed computer languages impose on users, but the inference that the solution is English, rather than a well-designed but nevertheless specialized language, is unwarranted even where generals or chairmen of boards are concerned. Even if English *is* involved, it should not be taken for granted that *both* the human user and the machine must use English. In fact, it may be most advantageous for the machine to use English while the person uses footpedals, buttons, pointings with lightpens, grunts, or other devices that linguists deprecatingly refer to as pre-linguistic. Along with McLuhan, I view such deprecations merely as snobberies induced by our culture based upon the printed book, "snobberies which ought to be cast aside."

A recent paper by Goodenough, for example, describes a lightpen-controlled program for on-line data analysis in which the computer displays a list of operator statements such as

CORR BETWEEN (X) AND (Y)

and a list of operands which Goodenough appropriately enough calls "menus." The user orders his computation by pointing with the lightpen. The machine takes care of all syntactic problems and displays operator and operand menus in proper alternation, and, while the user feels he is being addressed in English, he need never be bothered with details like whether "correlation" was spelled out or abbreviated with one "r" or two or whether the abbreviation was followed by a period or not.

Certainly the description of a complicated two- or three-dimensional object as in the systems of Sutherland, Roberts, or Jacks is not an easy matter in English. Consequently, designers of such systems have resorted

to elaborate network-like representations suitable for internal manipulation, such as the plexes of Ross and kindred devices, while reference by people is made through button-pushings, lightpen-pointings, and so on. There is occasional despair that the matter cannot be carried out in English by people, but I suspect it is better to pay close attention to the syntax of pre-linguistic modes of expression rather than to attempt to force-fit the arbitrary linear conventions of printing, which mirror the necessary basic linearity of speech but which are hardly essential when two- or higher-dimensional media are available. The description of which of a set of building blocks pushed together on a screen should be considered henceforth as a single entity would be odious in English, while the matter of drawing a line around them with the lightpen and pushing a button, setting off some kind of abbreviation mechanism in the internal representation system, is far more likely of success and is indeed representative of the lines along which current implementations are made.

Linguists and psychologists are, I think, quite unprepared for the implications on human communication of the effects of easily used two-dimensional dynamic representations. Nevertheless, snobbery should not blind us to the value, in a conversation between two people concerning which of a set of objects they mean, of substituting, for some elaborate oral description, coincident lightpen-pointings which brighten or set flashing the portion of a drawing referred to on a screen. I prefer to paraphrase McLuhan with the statement that "a visual image [of objects or concepts] has meanwhile been regaining its hold through electrical engineering."

In conclusion, I would like to cite one more statement of McLuhan, reflecting more aptly than anything I have seen in the computer literature the excitement and opportunity of the work in which we are engaged:

> Automation is information and it not only ends jobs in the world of work, it ends subjects in the world of learning. It does not end the world of learning. The future of work consists of earning a living in the automation age. This is a familiar pattern in electric technology in general. It ends the old dichotomies between culture and technology, between art and commerce, and between work and leisure. Whereas in the mechanical age of fragmentation leisure had been the absence of work, or mere idleness, the reverse is true in the electric age. As the age of information demands the simultaneous use of all our faculties, we discover that we are most at leisure when we are most intensely involved, very much as with the artists in all ages.

REFERENCES

HENLE, P. Mysticism and Semantics (in) Philosophy and Phenomenological Research quarterly, vol. IX (1948-49), pp. 416–22. Available: Department of Philosophy, State University of New York at Buffalo.

QUINE, W. V. O. 1961. *From a Logical Point of View.* Cambridge: Harvard University Press.

SESSION III

Languages and Communication

7

Problem-Oriented Languages for Man-Machine Communication in Engineering

STEVEN J. FENVES
University of Illinois

INTRODUCTION

The purpose of this paper is to describe some of the means for providing closer cooperation between men and computers. The paper is restricted to engineering applications, although the ideas and methods may be equally applicable to other areas, notably command-and-control and management systems.

Typically, in performing a design, an engineer starts with a loosely defined set of needs, criteria, concepts, and models and converges to a feasible and—hopefully—optimum solution. The conventional (batch) mode of computer use, immensely helpful in engineering analysis, is poorly suited for such design applications. First, the engineer dealing with a new problem must concern himself both with new procedures and new data. Thus, the conventional partitioning of problem-solving into programming and production is too artificial. When the use of previously prepared programs is attempted, it often results in stereotyping or "canning" of problem types and actually hampers the engineer's creativity. Second, the batch mode does not allow for the exercise of the unavoidable non-quantifiable decisions except at the expense of one decision per turn-around period. This lack of flexibility either forces wasted machine time and voluminous output to cover all possibilities or leads to searches, often futile, for "automated" design procedures based entirely on stored quantitative data.

Improvements in man-machine communication fall into two categories. The first category concerns the improvement in the rate of communication so as to reduce the turn-around time to minutes or seconds, such as provided by time-sharing and other direct-access facilities. However, time-sharing hardware *per se* is of little value to the engineer. Also needed is a second category of improvement concerned with raising the level of

communication essentially from the level of the machine to that of the user. Here, two approaches are available, namely, non-verbal communication, such as graphical or audio, and communication based on more suitable problem-oriented languages. An ideal man-machine system should incorporate both facilities, especially in engineering, where so much of the day-to-day communication is graphical.

Problem-oriented languages, however, being hardware-independent, have two points in their favor. First, from an economic point of view, they are evolutionary and can be used to advantage even if only the conventional mode is available. Second, from a philosophical standpoint, they possess an element of elegance, in that they use the logical capability of a general-purpose computer to transform itself into a special-purpose machine through appropriate programming, without the need for special-purpose devices such as lightpens or special keyboards.

DEFINITION OF PROBLEM-ORIENTED LANGUAGES

The most difficult problem concerning problem-oriented languages, or POL's, is to find a suitable definition. This is due to the simple fact that different people have different problems and therefore need different levels of languages. As a simple illustration, consider the problem of computing an integral by the trapezoidal rule. A programmer, charged with the task of devising a detailed algorithm, will find any algebraic procedural language suitable and could write, for example (in FORTRAN II):

```
      .
      .
      .
      DIMENSION Y(100)
      SUM = (Y(1) + Y(N))/2.
      M = N-1
      DO 1 I = 2,M,1
    1 SUM = SUM + Y(I)
      A = H*SUM
      .
      .
      .
```

A numerical analyst who has an integral to evaluate as a part of a larger problem may find a closed subroutine to be the most closely "problem-oriented" and would prefer to simply write:

```
A = TRAPEZ(Y,N,H)
```

A working definition of a POL may be the following: *A language is problem-oriented if it relieves the programmer from the necessity of specifying all the details which the associated processor can automatically provide.*

In the field of engineering, at least, a POL and its associated processor can best be defined in terms of its desirable attributes, which may be summarized as follows:

1. A unique program is written for each new problem. This fundamental property of a POL implies that there is no distinction between procedures and data and recognizes that new solutions are obtained by applying one's accumulated knowledge in novel combinations.
2. The source program is meaningful to the user in the sense that it is its own document in the user's language.
3. The POL contains provisions for exercising judgment. It permits ease of change or modification of the problem being worked on, and the output provides meaningful measures of the consequences of a decision.
4. No constraints on input format. The language is designed with the engineer, not the keypunch operator, in mind.
5. No constraints on problem size or complexity. The processor should have none of the irritating restrictions, inevitably due to unimaginative programming, which so often plague canned programs.
6. Efficiency over entire range of problems. Execution of small or simple problems should not be penalized by the facility for doing large or complex jobs.
7. Expandable scope. The language and its associated processor must provide for ease in expansion of the vocabulary and scope.

The points raised above will be further discussed and exemplified in the remainder of this paper. An illustrative example of a relatively simple system which has the first four attributes is taken from the University of Illinois Statistical Service Unit Library.* The following is the complete specification for a linear programming problem:

```
LINEAR PROGRAM FOR (1) PROBLEM.
MAXIMIZE THE FUNCTION = .5X(1) + 6X(2) + 5X(3),
CONSTRAINT (1) 4X(1) + 6X(2) + 3X(3) LE 24,
CONSTRAINT (2)  X(1) + 1.5X(2) + 3X(3) LE 12,
CONSTRAINT (3) 3X(1) + x(2) LE 12,
END OF PROBLEM.
```

Admittedly, the vocabulary is limited (the only other acceptable words are MINIMIZE and the MAD-FORTRAN IV logical connectives). Within this context, however, the language is readable and meaningful and requires no instruction.

Before going into details, a comment about input formats seems to be in order. There exists today a tremendous disparity between programmers' languages (source programs) and users' languages (data). The FORTRAN programmer takes full advantage of the free-field facility of *his* source language, but he does not pass on his labor-saving devices

* "Manual of Computer Programs for Statistical Analysis," Statistical Service Unit, University of Illinois, Urbana, Illinois, 1964.

to the user, who is constrained to an absolute fixed format, with disastrous results if he trespasses. Thus, a FORTRAN programmer may freely write two similar statements in entirely different formats, e.g.:

```
READbbbINPUTbbbTAPEb7,bb100bb,I,bbA
RIT7,100,J,B
```

but if he uses

```
100  FORMAT(I3,F 8.2)
```

the user must fill in his typical data as

```
b12bbb23.45
bb3b1234.00
```

and a card punched as

```
bb12bb23.45
```

will immediately cause a fatal error in the input routine.

The consequence of this inequity is that for many users the computer means only the drudgery of filling out dreary input forms. This approach can only lead to a degradation of the engineer's work. A POL designed to be used by engineers must eliminate this feature and allow for completely free-field input, even at the expense of slightly longer running times.

THE COMPONENTS OF A POL

In the remaining discussion, the system consisting of both the source language and the associated processor will be considered as an entity. Thus, the system consists essentially of three components:

1. The language itself, in which the problem is communicated;
2. The application programs, or subroutines, which perform pieces of execution directed by or implied in the language;
3. The systems program which exercises control over the entire process.

Although these three components are closely interrelated, it is profitable to examine each of them separately.

The Source Language

The specification of the input language for a POL is a major engineering design job. It requires a thorough analysis of the computational, logical, and data-processing operations within the discipline for which the POL is being created to ascertain what is actually being done and how these operations are interpreted in the profession.

Fortunately, the specification of a language for an engineering discipline is facilitated by three factors. First, any one engineering discipline deals with only a relatively small number of distinct computational processes. All problems can then be formulated as an appropriate, and not necessarily unique, combination of these basic processes. As an example, an extremely comprehensive POL in hydraulic engineering being developed is to contain an estimated 137 basic processes.* This number is probably quite high in comparison with the working repertoire of most practicing hydraulic engineers.

Second, each of the basic processes has an accepted name within the discipline, used in everyday communication between engineers. Adapting this terminology as the input language makes it thus possible to "squeeze" into a word the information content of hundreds of algebraic and logical statements. Of course, this engineering language has to be augmented by a few computer-oriented words. It also must be admitted that we are concerned here with a highly professional language, acquired after 16 to 20 years of formal schooling, and that the same conciseness may not apply elsewhere.

As an example, the table below shows an abbreviated list of the available words (commands) in COGO (COordinate GeOmetry),** a POL for plane geometry computations in surveying.

ADJUST/AZIMUTH	LOCATE/AZIMUTH
ANGLE	OFFSET/POINT/LINE
AREA	PARALLEL/LINE
AZIMUTH/INTERSECT	PAUSE
CLEAR	POINTS/INTERSECT
CURVE/LINE/AZIMUTH	ROTATE/TRANSLATE
DEFLECTION	SEGMENT
DUMP	SIMPLE/CURVE
EQUIDIVIDE/LINE	STOP
FIT/CURVE	STORE
INVERSE/AZIMUTH	SUBDIVIDE/LINE

It can be seen that the words reflect closely the surveyor's operations, such as ADJUSTing traverses and computing AREAS. The four computer-oriented commands shown (DUMP, PAUSE, STOP, and STORE) are also self-explanatory. With this vocabulary, problems in surveying can be quickly and conveniently built up.

* G. Bugliarello, "Toward a Computer Language for Hydraulic Engineering," Proceedings, IX General Meeting, International Assn. for Hydraulic Research, Belgrade, 1961.

** "COGO1—A Programming System for Civil Engineering Problems," IBM 1620 Program Library, Program No. 1620-UG-01X, IBM Corp., White Plains, N. Y.

For example, in Figure 1, given the coordinates of point 1, the length and azimuth (clockwise angle from north) of lines 1-7 and 1-95, it is required to compute the coordinates of points 7 and 95 and the area of the triangle. The COGO program is as follows:

```
STORE            1        1000.     2000.
LOCATE/AZIMUTH   7   1    256.17    45   00   00
LOCATE/AZIMUTH   95  1    350.0     102  35   12.35
AREA             1   7    95   1
PAUSE
```

The second line above reads: *Locate* point *7* by going from point *1* a distance of *256.17* at an *azimuth* of *45* degrees *00* minutes *00* seconds. The remaining lines are equally self-explanatory.

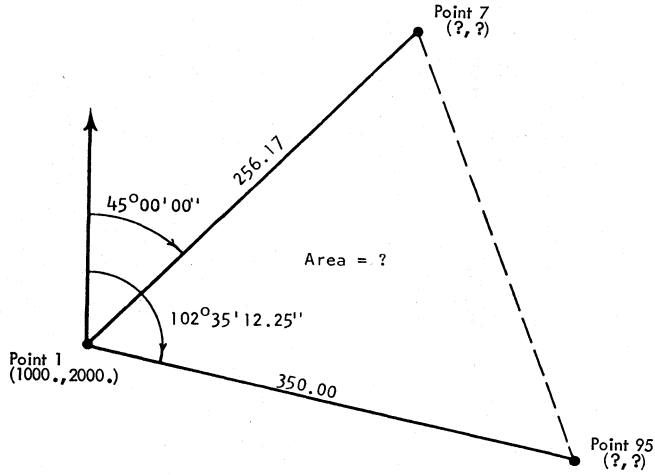


FIGURE 1. Sample COGO problem

The third factor facilitating the selection of the language is the fact that a well-designed POL allows for an open-ended repertoire of basic processes so that new terms and their corresponding execution routines can easily be inserted. This is a feature that seems somewhat novel to professional systems programmers, who are accustomed to having the entire system under their own control and who like to deal with relatively stable systems. However, this flexibility is essential, both because it reflects the evolutionary nature of engineering and because it is the only way to overcome the restriction of scope that is unavoidable in any pro-

gramming language. As an indication of the dynamic nature of POL's, organizations making heavy use of COGO report alternations between periods of expansion of the language, when new commands are added at a rapid rate, and periods of consolidation, when redundant or seldom-used commands are removed from the system. Thus the system becomes a dynamic entity, much like a natural language.

Applications Programming

In the simplest sense, the execution program of a POL can be thought of as a series of subroutines in a one-to-one correspondence with the language elements. In practice, of course, the system is considerably more complex, and extensive chaining and logical cross-connecting must be established to achieve the desired generality and flexibility.

Just as the design of the language, the development of the execution subroutines must be the responsibility of engineers engaged in the discipline for which the POL is being written. First, the engineers understand the underlying mathematical or physical model with its inherent assumptions and limitations. Second, the generality, the flexibility, and especially the efficiency needed for a large-scale system come from a formulation based on thorough understanding of the problem, and not from clever coding of a mediocre formulation.

The flexibility and the efficiency that result from a general formulation are best illustrated by STRESS (STRuctural Engineering Systems Solver), a POL written for the analysis of elastic structures.* Over the years, structural engineers have evolved many special-purpose algorithms for analyzing various types of structures. Many of these algorithms have been programmed and used extensively both for analysis and for design. However, in developing STRESS, the decision was made to formulate a single procedure which encompasses the majority, if not all, of the special methods.

The present version of STRESS is restricted to the static analysis of elastic framed structures, but efforts are under way to extend the capabilities in several directions. The description of a problem in STRESS consists of procedure descriptors (such as the method of analysis to be used, type of structure, kinds of output desired), size descriptors, and data descriptors.

The three types of data (geometry, topology, and mechanical properties) describing the structure are carefully separated. Furthermore, in describing the member properties, the word PRISMATIC is used to designate the procedure to be used by the program in converting the input data (in this

* S. J. Fenves, R. D. Logcher, S. P. Mauch, and K. R. Reinschmidt, "STRESS—A User's Manual," Cambridge, Mass., M. I. T. Press, 1964.

case, the cross-sectional area A_x) into the required properties (in this case, the stiffness coefficient of the bar). As an example, the STRESS program for analysis of the space truss shown in Figure 2 is given on left below.

```
STRUCTURE SAMPLE TOWER
TYPE SPACE TRUSS
NUMBER OF JOINTS 7
NUMBER OF SUPPORTS 3
NUMBER OF MEMBERS 12
METHOD STIFFNESS
JOINT COORDINATES
1 X 0.0 Y 16.0 Z  0.0
2 X 0.0 Y  8.0 Z  0.0
3 X 4.0 Y  8.0 Z  4.0
4 X 4.0 Y  8.0 Z -4.0
5 X 0.0 Y  0.0 Z  0.0 SUPPORT
6 X 8.0 Y  0.0 Z  8.0 SUPPORT
7 X 8.0 Y  0.0 Z -8.0 SUPPORT
MEMBER INCIDENCES
1 2 1
 2 3 1
 3 4 1
 4 2 3
 5 3 4
 6 4 2
 7 5 2
 8 5 3
 9 6 3
10 6 4
11 7 4
12 7 2
MEMBER PROPERTIES, PRISMATIC
 1 AX   1.0
 2 AX   1.0
 3 AX   1.0
 4 AX   1.0
 5 AX   1.0
 6 AX   1.0
 7 AX   1.0
 8 AX   1.0
 9 AX   1.0
10 AX   1.0
11 AX   1.0
12 AX   1.0
NUMBER OF LOADINGS 1
LOADING ARBITRARY
TABULATE FORCES, JOINT DISPLACEMENTS, REACTIONS
JOINT LOADS
1 FORCE Y -10.0, FORCE Z 5.0
4 FORCE X  6.0
SOLVE THIS PART
```

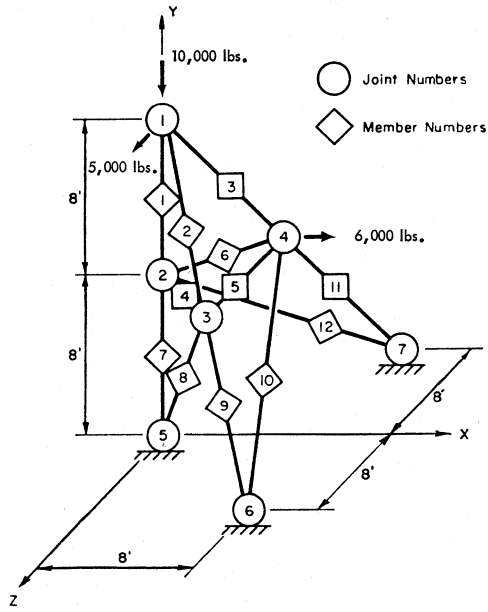


FIGURE 2. Sample STRESS problem

The advantage of such an approach is not just that any structure can be conveniently analyzed, but that any portion of the structure can be singled out for modification by the STRESS-vocabulary words ADDITIONS, DELETIONS, and CHANGES. This facility, especially in a time-shared environment, brings us quite close to the crucial attribute mentioned earlier, namely, that the system output provides meaningful measures of the engineer's decisions.

Systems Programming

The third ingredient of a POL is the system or control program which ties the previous two components together. Here we begin to get out of the usual area of competence of engineers and must rely heavily on work done in systems programming and other areas of applications. Unfortunately, it appears that in the past the needs of POL's have always been a few steps ahead of current state-of-art, and most POL's in existence have been designed and coded entirely by applications programmers.

There are essentially two components of systems programming involved. The first one deals with input processing and decoding. This phase obviously bears close resemblance to the first pass of any compiler. In most cases, the source language is actually simpler in structure than a procedural language. On the other hand, the data structures generated are often more complex than in the present algebraic procedural languages.

The second component of the control program deals with the execution control, where the actual execution is performed by precoded subroutines. There are several major constraints on this phase of the problem. First, as mentioned earlier, efficiency over the entire range of problem sizes is a major consideration. This overriding consideration requires a usable dynamic memory allocation scheme, both for the complex data structures involved and for the hierarchically arranged subroutines. Second, the system must adjust to the fact that the execution subroutines will be written by average applications programmers, and not by systems programmers. Thus, many of the system functions, such as the dynamic memory allocation process, must be performed by the control program. And third, the time-shared environment imposes additional considerations which must be satisfactorily dealt with.

POL's in existence today employ a wide range of implementations, ranging from preprocessors through interpreters and assemblers to compilers. The control system for COGO (Figure 3) is essentially a pure interpreter. This much-maligned word has to be taken in its most general sense, however. Specifically, in COGO, about 25 machine instructions are needed to interpret a statement, but some of these can cause several hundred FORTRAN statements to be executed. The simplicity of the COGO control program comes from the fact that its language is essentially context-free and that its commands communicate with each other only through the table of coordinates.

On the other hand, the control program of STRESS is considerably more complex and consists of an input decoder and syntax checker, a semantics checker, and a sequence of execution drivers (Figure 4). The boxes marked with an asterisk show the parts of the program omitted in the time-shared version. This illustrates a point which came as a surprise

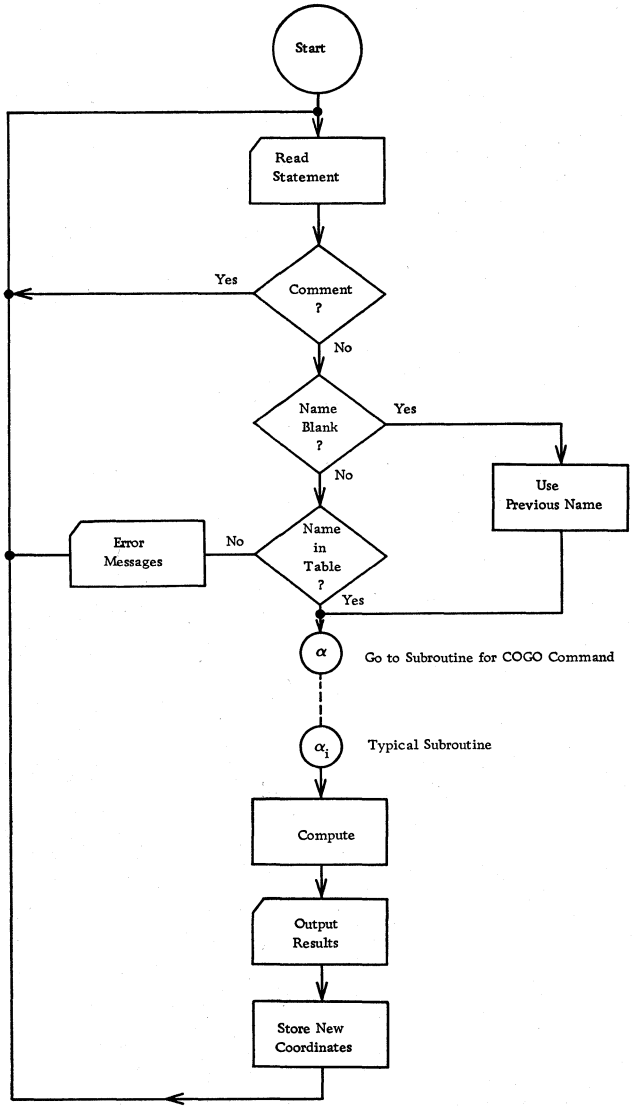


FIGURE 3. Simplified flow diagram of COGO

to us: namely, that it is actually simpler to write an on-line control system than an off-line one, since the user at the console can immediately take action if an error is detected.

Because of the open-ended nature of POL's, we have come to the conclusion that it is a hopeless task to keep the system up to date and

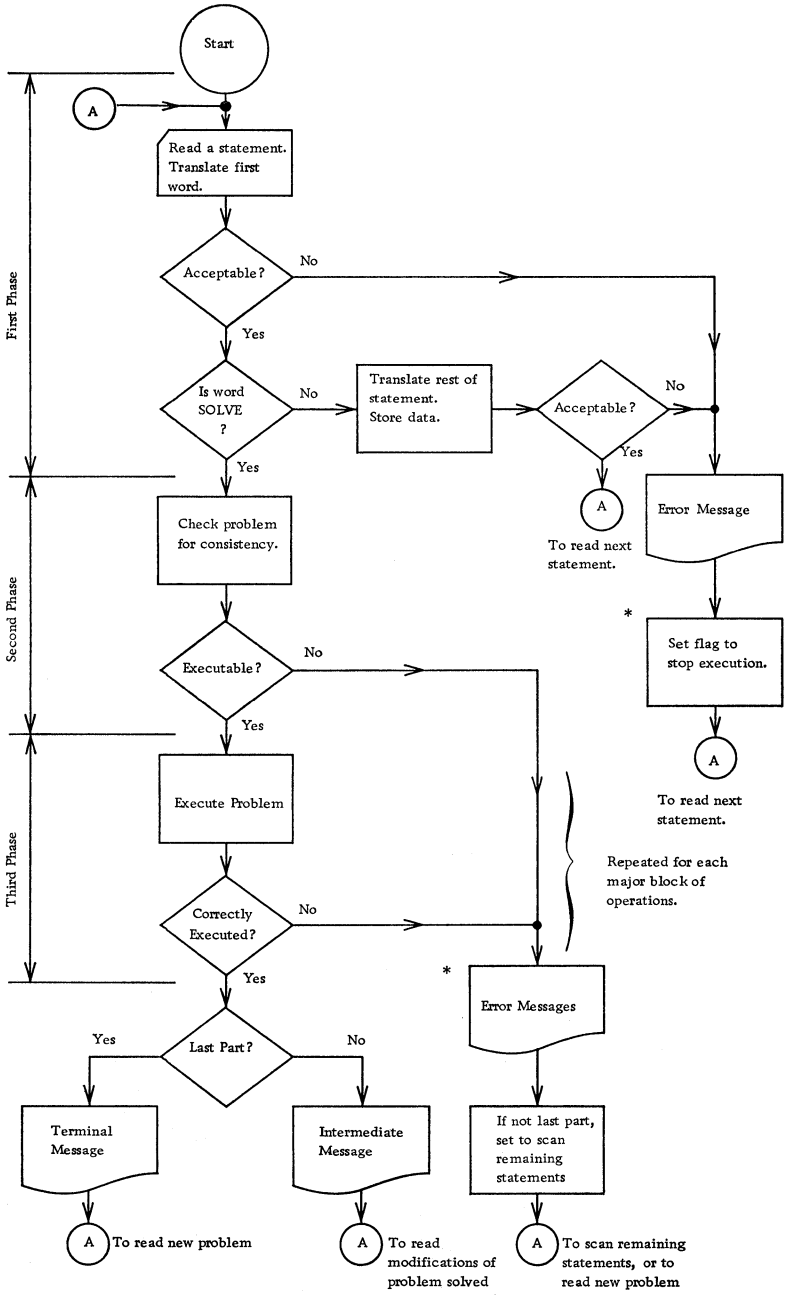


FIGURE 4. Simplified flow diagram of STRESS

in pace with the emerging additions. An ideal framework for a dynamically expanding POL would be one where the generation of the system itself is mechanized. We are currently working on such a system for STRESS-like languages (Figure 5), consisting of a translator-generator which produces the input decoder from a formal description of the source language and a preprocessor which will convert subroutines written in a procedural language into equivalent subroutines compatible with the internal data organization.

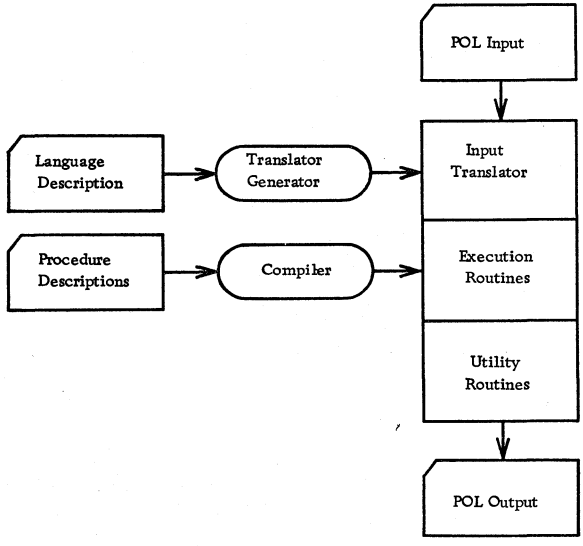


FIGURE 5. Proposed POL system

CONCLUSIONS

In conclusion, two topics should be briefly mentioned. One is the fear expressed by some that the proliferation of languages will be more harmful than beneficial. I believe that this fear is largely unfounded. First, if the language is truly problem-oriented, there need be no distinction whatsoever between learning one's profession and learning its associated POL. Second, with a general system such as the one mentioned above, it should be a straightforward process to merge two or more POL's into a common frame of reference, should a need arise.

The second topic concerns the future of engineering. It is becoming clear that the combination of time-sharing and POL's will have a revolutionary effect on engineering and will make it the challenging, creative profession that it ought to be. Rather than being burdened with lengthy

calculations and routine decisions, the engineer will be able to concentrate on his problem and will have access to all the procedures and data which have relevance to his task. Thus, the computer will work as a partner by properly matching its best capabilities to the creative and synthesizing abilities of the engineer. The opportunity and challenge of this combination will make the world an exciting place for all of us.

DISCUSSION

F. G. WITHINGTON: I have a two-part question, based on Professor Oettinger's paper (paper 6). One, do you expect that graphic outputs will increase in importance in those disciplines with which you are familiar? Two, if so, will lightpens, buttons, or "grunts" tend to become part of the problem-oriented language?

S. J. FENVES: There is a very close interrelationship between graphical language and problem-oriented languages. As a matter of fact, at M. I. T., both COGO and STRESS have been combined with SKETCHPAD, and it is obvious that such interconnections will become more frequent. As far as button pushing is concerned, I am a little bit concerned. Unless there is an easily implemented mechanism to record internally the settings of the switches or the sequence of buttons pushed, I would hesitate to involve in a complex procedure anything that cannot later be documented and retrieved.

H. W. BROUGH: In this process with COGO that you describe, of adding to and then consolidating the language, does the language permit inclusion of new syntax by nesting or compounding the existing syntax? If so, does the language include iteration?

S. J. FENVES: Yes, the system is built in this fashion. Most of these changes are made by nesting sequences of individual operations and giving them a name, but even some of the basic parts of the system are nested. In COGO, the first part of the traverse adjustment routine is simply an iteration on the LOCATE routine. Thus, internally the system uses nesting and iteration, and changing the nesting is very simply done.

H. KRENN: Apparently your system consists of two parts: the control program and the application programs. How independent is the control program from the application programs? Would it, for instance, be possible to use the same control program in connection with completely different sets of application programs, and would this control program thus process completely different languages?

S. J. FENVES: This depends upon the complexity of the problem. COGO is essentially a context-free language where no information gets passed from one statement to another except the data in the common area containing the coordinates of points previously defined and where

all routines either create new coordinates or operate on previous coordinates. If you have a similar problem, then you can take the fifty-word COGO control routine and write an entirely new set of operations.

This is not quite true of STRESS, but, within the present framework of STRESS, any network-type operation can be performed. We have not programmed electrical networks because it is inefficient to handle one-dimensional networks with a mechanism that is built entirely for variably-dimensioned networks, but this could easily be included. Also, we are looking into handling CPM (Critical Path Method) and other operational networks within the same framework as structural analysis.

8
Some Methods of Graphical Debugging*

THOMAS G. STOCKHAM, JR.
Massachusetts Institute of Technology

In the days of the cathode-ray electrostatic storage tube, computer users were aware of the fact that they could often extract important information about the behavior of their programs by observing the luminescent patterns produced on the storage grid by the read-write scan. Perhaps this was the earliest form of graphical program analysis. Debugging techniques based upon this principle did not evolve into common practice for a variety of technical and economic reasons. However, over the years a few modest experiments in on-line program analysis using similar ideas have been performed. As computational power becomes more accessible and the equipment for man-machine interaction continues to evolve, a re-awakening of the techniques exposed by these experiments is being stimulated. In this paper we discuss some of these experiments and then suggest methods for reviving their techniques in a modern atmosphere.

Undoubtedly the first experiments in program analysis that involved graphical methods were those involving high-speed core dumps via displays. Programs such as these were well known at installations with the appropriate facilities as early as ten years ago. Certainly it can be argued that strictly speaking there is very little graphical about a simple display dump, but we mention it here since its speed and flexibility are characteristic of important graphical debugging methods. Also, displayed text is a basic building block in most graphical debugging schemes. The author's own version of a core display was written for the TX-0 computer in

* Work reported herein was supported in part by Project MAC, an M. I. T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research contract number Nonr-4102(01). Reproduction in whole or in part is permitted for any purpose of the United States government.

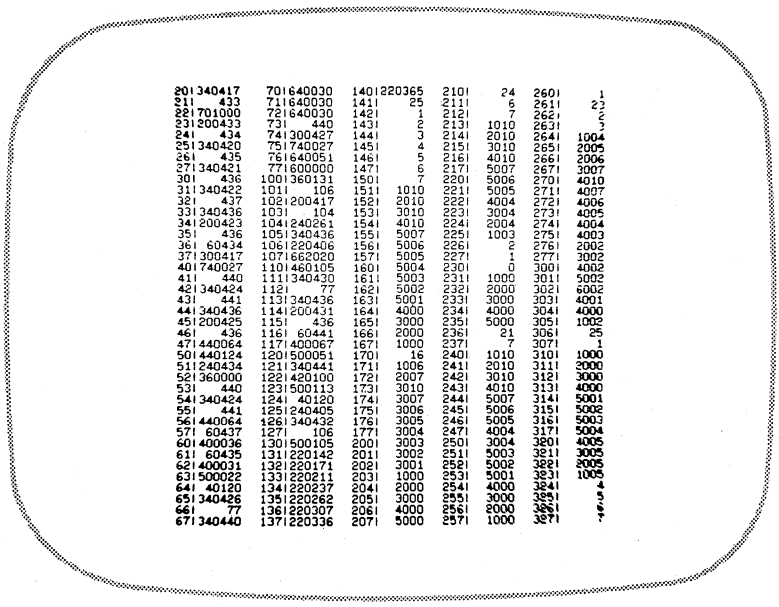


FIGURE 1

[This and the following eight other illustrations are much-reduced rough facsimiles, reproduced from photographic "snapshots" of actual graphic displays. The original displays not only were much larger, but also had greater clarity and resolution of detail. The grey frame represents the cathode-tube medium of visual output.]

September 1958. Some typical output of this program is presented in Figure 1. This primitive octal dump used a point display and required about five seconds for generation. The present state-of-the-art in computer displays permits the display of this quantity of text on the order of 100 milliseconds. There has been one main lesson learned from using core displays such as these. It is that when engaged in on-line debugging other more conventional methods of program communication are greatly reinforced by the insight and confidence-building impressions that can be obtained from a swift perusal of large arrays of data or from the rapid redisplay of smaller arrays in a dynamic state.

Another experiment which used the basic core-display idea to place in evidence the dynamic patterns of a program in operation was performed on the TX-0 computer in March 1959. A similar endeavor called MEMORY COURSE was reported by Licklider and Clark in 1962.[1] A program was

written to perform a full interpretation of any TX-0 code and to display all machine conditions after the completion of each instruction. The machine conditions included the states of the accumulator, the live register (similar to a multiplier-quotient register), the index register, and the program counter. The contents of the program counter were displayed at scope locations which corresponded to their numerical values, and thus a pattern similar to those formerly achieved on storage tubes was obtained. In addition, a few instructions at and in the vicinity of the program counter location were displayed. At that time symbolic debugging was virtually unknown, and all quantities were displayed as octal equivalents. Figure 2 shows the program being used to analyze the display dump program discussed earlier.

Two important features of this program were the abilities to assign a breakpoint to any core location and to inhibit display except at instructions to which the breakpoint was assigned. In this manner it was possible to run a program at full interpretive speed and to strobe the machine conditions at specific program locations. The effect was very roughly the same as that obtained when stroboscopic light is used to slow or stop mechanical motion. By strobing the accumulator, it was possible

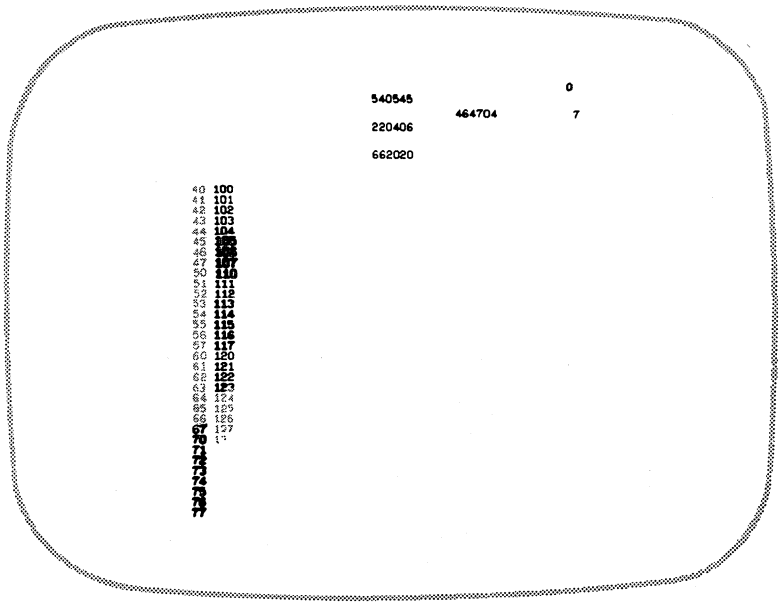


FIGURE 2

for one user to spot some peculiar arithmetic faults in the middle of a double-precision subroutine as it scanned a table of data. These faults had gone unnoticed during design checkouts. The most common and a very powerful use of this feature was in following the state of a particular program variable responsible for subsequent endless looping or faulty conditioning. In a majority of situations, it proved quite simple to watch a speeding index count and to sneak up on a faulty phase of logic without paying the price of a trace or of stepping through each new variable state at manual rates.

An experiment called PROGRAM GRAPH, which carried the idea of displaying program parameters in a slightly different direction, was implemented by Licklider and Clark[1] for the PDP-1 computer. This program was designed to display program parameters versus time as conventional graphs. Later I. E. Sutherland modified this attempt for use on the TX-2 computer by concentrating on displays of the program counter. The resulting plotted function was composed of segments of straight lines. Each discontinuity corresponded to some form of transfer instruction. Figure 3 shows typical output from this display. The program under test contained one loop which called a single subroutine containing no loops

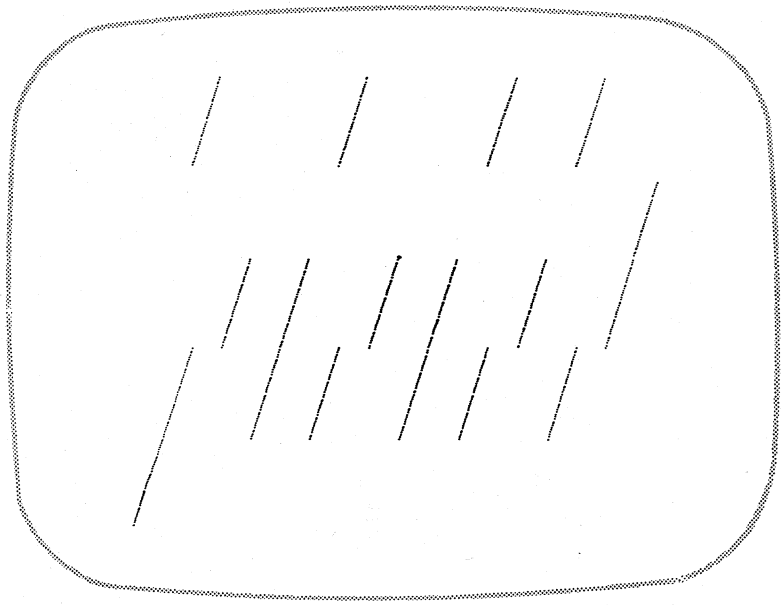


FIGURE 3

and was run periodically to create a more or less continuous display. In addition, by varying the parameters of the program under test, the various patterns of flow could be sampled and compared with the intended logic in a convenient, rapid manner.

An important characteristic of Sutherland's experiment was that the program achieved its end without having to interpret the program under analysis. This was accomplished by multi-programming methods which effectively forced an interrupt after each analyzed instruction. The time savings implied are important, especially when considering a widespread use of any kind of dynamic program analysis.

In recent years some attempts to deduce program-control flow from source-language statements and object codes have met with encouraging success. Among these are the efforts of L. M. Haibt[2] of IBM, P. Shermann of Bell Telephone Laboratories, and W. R. Sutherland of the Lincoln Laboratory. Sutherland's efforts, completed in 1963, produce a flow-chart display from TX-2 object code with auxiliary information supplied by the translator. Figure 4 shows some typical output. Seven seconds were required to produce this display, which corresponds to forty machine-language statements. Of the seven seconds, roughly one second was

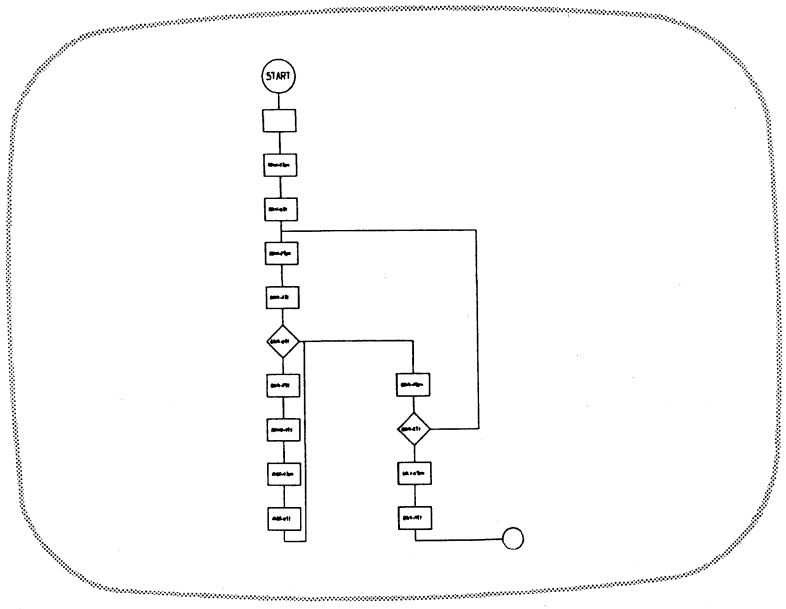


FIGURE 4

spent performing the actual flow analysis. The rest of the time was required to create the actual displayed image via the drawing facilities of the SKETCHPAD program of I. E. Sutherland,[3] which does not make use of recent advances in line and curve drawing display technology.

A most important point involved here is that the data which is produced by the flow analysis program and which is used to create the flow-chart drawing is in the form of a list-structure. This circumstance provides several important facilities. First, only part of the structure need be expanded into the final drawing, and this may be done for different parts of the program without repeating the basic analysis. Second, the structure places in evidence the actual topology of the associated program. Third, the structure permits a convenient path for associating any demonstrative action that may be taken in reference to the flow-chart drawing with the corresponding portion of the analyzed program. The latter point will become most important in our subsequent discussion.

In October 1964 the author completed work on a displaying text-editing program for the TX-2 on-line programming system. This facility allows the user to select up to two or three dozen lines of source-language program for display at one time and provides for convenient insertion of characters at or near a moveable marker-character or cursor. In order to minimize the time required to effect changes, the characters are strung in a threaded list, using a separate list element for each character.

An important feature of a displaying edit program is that it allows faster cursor positioning than conventional methods. During any edit, a search for position requires a time proportional to the length of the text unless that text is placed in a list-structure to facilitate searching. To add to the problem, people always want some form of context search to help them compute a position that they could point to if they could see the text. This constitutes a somewhat inappropriate use of context searching and multiplies linear search times by factors roughly proportional to the frequency of the first character of the context string and to the relative inefficiency of a more general searching algorithm. In a time-shared system where one expects to find a large quantity of editing taking place, such inefficiencies should be eliminated. A displaying edit simplifies the problem in one of two ways.

When a lightpen or similar device can be used to sense the time at which a specific character is displayed, positioning may be achieved by interrupting the data channel at the time and examining its position in the output table. The corresponding list element can then be found in a list-pointer table. This method really uses the channel processor to perform the required linear search computation.

When a lightpen is not available, the user can serve as a comparator in a tree-like search for position by line, word, and character. By a word

we mean a group of characters separated by blanks, tabs, etc. Even if the text is not structured, this method is fast, since end-of-line and end-of-word tests are fast compared with the multi-character comparison algorithms required by identifier or context string searches. What is more, it is very simple to structure the text by line, word, and character when it is threaded for editing.

Out of these experiments, one can select five important characterizing techniques or philosophies. They are as follows:

I. The display of the dynamics of program state
II. The display of topological information about program-control flow
III. The maintenance of complete control over a program under dynamic analysis without an unreasonable increase in time required to run that program
IV. The graphical association of breakpoints and the strobing of the corresponding program states
V. The plotting of program variables as conventional graphs

It is the present objective to discuss the organization of these techniques in a modern atmosphere. As a matter of review, let us first examine a list of the most important on-line debugging techniques that are enjoying popular use today.[4,5,6,7,8,9]

1. Fully symbolic two-way communications between the programmer and the program under test[6,7,8,9]
2. Interrogation of program variables in a variety of formats[4,5,6,7,8,9]
3. Modification of program variables, constants, and instructions[4,5,6,7,8] or statements[9]
4. Facilities for preserving program state to facilitate back-tracking
5. Automatic comparison of program states for the detection of state changes[4,5,6,7,8,9]
6. Editing including word and context searching[10,11]
7. Breakpoint[6,7,8,9] or trace testing

A blend of these techniques might take a variety of forms in the production of a suitable graphical debugging package. Efforts to this end at Project MAC take a form implied in the hypothetical example which follows.

After a first attempt at running a program, a user discovers that a fault has occurred somewhere in the early part of his code. He then associates a breakpoint with one of his early program statements and initiates a second run. After several successful commands to reposition the breakpoint and proceed, the programmer learns that control is proceeding logically through a modest section of commands. Finally, having issued

a final command to reposition and proceed, the programmer finds that some uncontrolled action has ensued, and he stops the run. Now he calls for a graphical debugging feature to be known as FLOW-MAPPING. FLOW-MAPS are like flow-charts except that they are derived from actual control flow at run time. The programmer commands that when control first reaches the last successful breakpoint that he found above, a FLOW-MAP be constructed for $\frac{1}{10}$ second of computation time or until the 10th new transfer of control takes place. Again he starts his program from a fresh beginning. Soon a picture as shown in Figure 5 appears on his display, placing in evidence that portion of the program that was reached by the program flow after it passed into an uncontrolled state. After a few moments of examination, the programmer decides that something is not completely right with a certain area of the code as represented by the FLOW-MAP, and he makes the enlargement of Figure 6 in order to further his theory. Finally, the enlargement to Figure 7 is made when the conditional computation on the left is felt to be most suspect.

The mechanism needed for producing FLOW-MAPS is relatively simple. First the computer is placed in a mode where a testing supervisor can oversee the code under test and detect when control first passes the

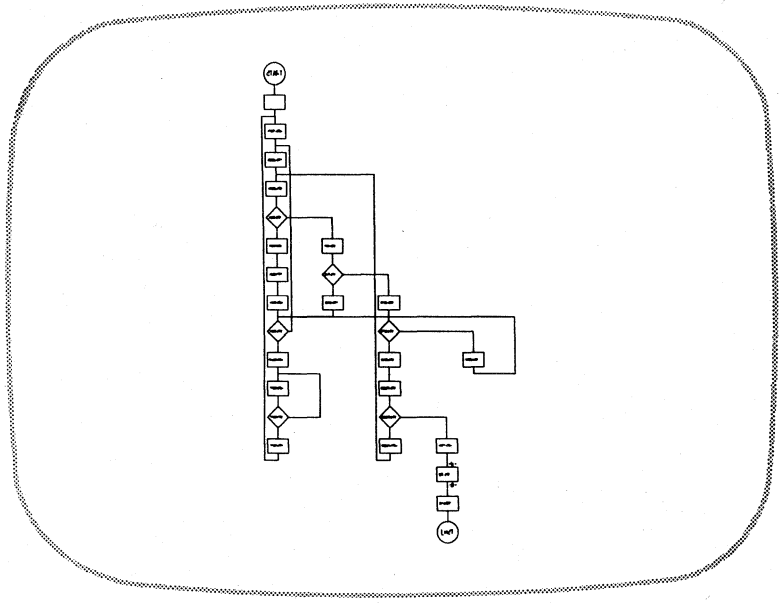


FIGURE 5

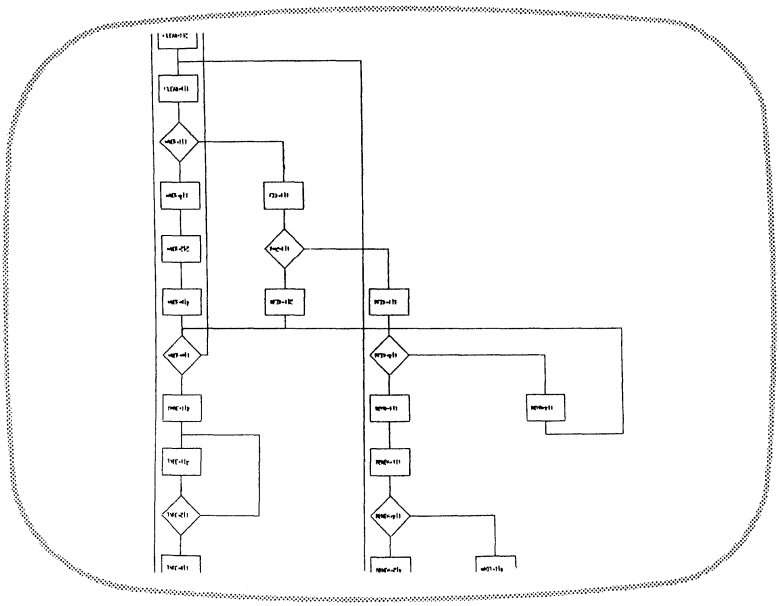


FIGURE 6

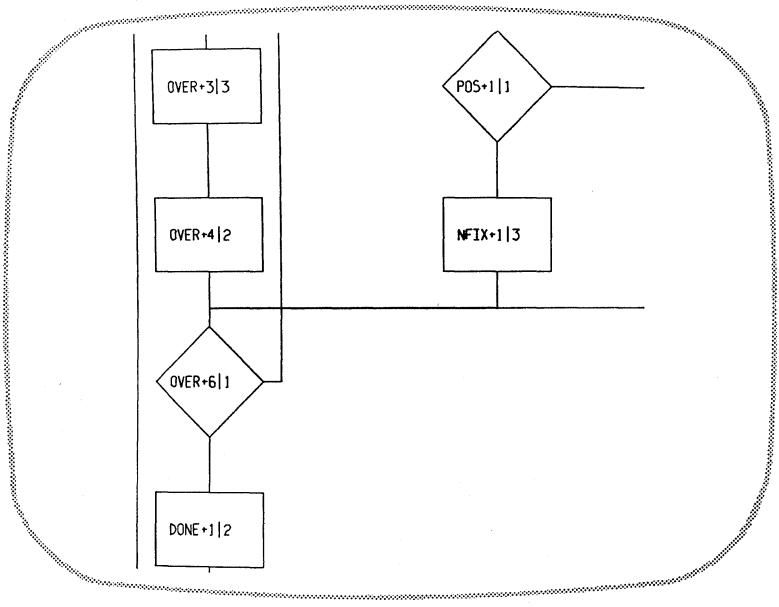


FIGURE 7

critical point. Then all transfers of control are supervised and reported to a portion of the debugging facility which builds a list-structure in analogy with the flow-path topology. This list-structure grows rapidly at first and then more slowly as control is passed to completely new code less and less often. After the allotted time this process stops, and displays (such as Figures 5, 6, and 7) are produced from the resulting list-structure. The testing supervisor mentioned above need never be conceptually more elaborate than a machine-language interpreter, but it is important that appropriate hardware be used instead, for speed reasons.

Next the programmer decides that a program variable, specifically the contents of index register 2, is being treated improperly during the conditional computation mentioned above. With a lightpen he points to a flow line leaving that decision element and presses a button allocated to the selection of a data probe. After the symbolic name of index register 2 is typed, the display of Figure 8 is produced. Thereafter, whenever control passes the selected point, the display of the probed variable is adjusted to agree with its current value. Breakpoints would be associated with the code under test in a similar manner.

The form in which probed data is displayed can of course be varied to suit any standard format. In some cases it might be accumulated for display in graphical form. Such a display is shown in Figure 9. This example is typical of the history of a variable associated with a numerical computation. Programming faults often show up as marked discontinuities or as a similar lack of smoothness in plots of numeric variables.

The mechanism for graphically associating a data probe or a breakpoint is not as complex as it might seem. When the lightpen sees the flow line in question, the display channel is interrupted, and the program notes the member of the FLOW-MAP list-structure that is responsible for that particular display item. Since that list-structure member was created with a knowledge of program flow location, that information can be and is stored as part of the data contained in the member. When the button for selecting a data probe is pressed and the lightpen has singled out a FLOW-MAP list-structure member, that member is examined for the associated program location, and the hardware is adjusted to trap to the data-probing procedure whenever control reaches that location. Also, the FLOW-MAP display is augmented to include the data-probe symbol needed to provide suitable communication to the user as to its presence and data assignment.

After the program is allowed to run with the data probe attached, it is observed that index register 2 is indeed responsible for the faulty behavior. Knowing this, the user desires to modify his source-language program. To effect this, he could point his lightpen at a flow box above the conditional computation and press a button allocated to the initiation

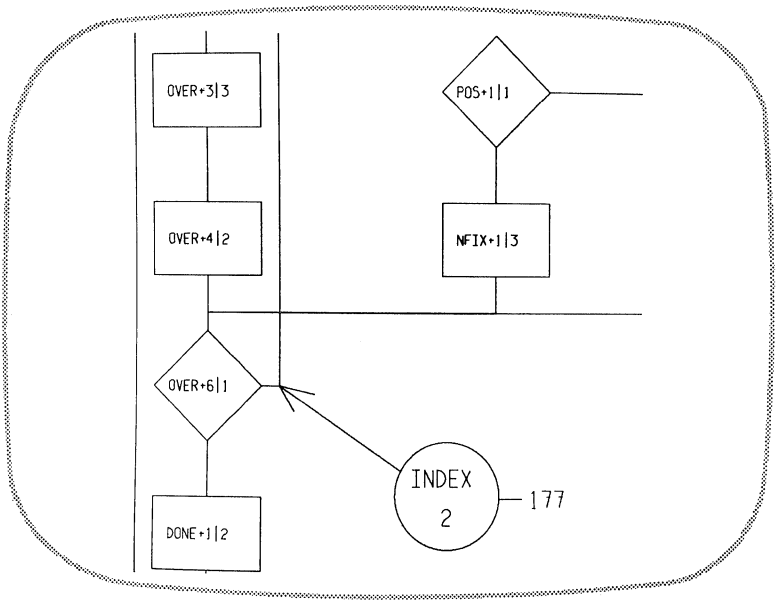


FIGURE 8

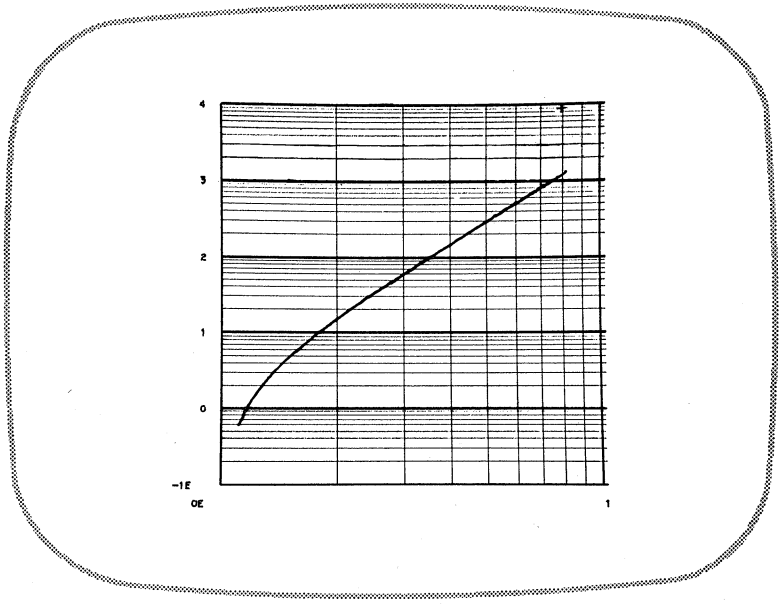


FIGURE 9

of the display of source-language text. Soon those statements corresponding to the flow box selected are displayed, and subsequent depressions of another button cause successive sections of text to be selected for display. When the faulty statements are displayed, they may be edited directly. After a suitable complete or partial recompilation, the analysis processes begin again.

There are some interesting prospects which are a by-product of this discussion. The association of input and output statements with a given procedure almost always presents a difficult consideration in programming. It is an interesting thought to contemplate inserting these aspects of a procedure graphically via a FLOW-MAP or flow-chart after a basic algorithm is designed. The basic advantages envisioned are flexibility and simplicity. If it is easy to explore the state of a variable that would not ordinarily be explored, better understanding of program performance is the result. Also, there would be a certain elegance gained by being able to divorce the input-output phases of programming from the purely algorithmic ones.

The thought of being able to specify a portion of a procedure by graphical means naturally leads to the idea of specifying entire procedures in that manner. The concept of a graphical compiler (i.e., a compiler which uses pictures as a source language) is not a new one. For instance, people have been thinking of compiling flow-charts for some time and have made some primitive attempts in this direction. Efforts to this end are now in progress at the Lincoln Laboratory and at The RAND Corporation. As these efforts mature and evolve, some processes of program specifications, program maintenance, and program debugging will tend to take place in a common language that in all probability will enrich and facilitate all three.

REFERENCES

[1] J. C. R. LICKLIDER and W. E. CLARK, *On-Line Man-Computer Communication*, Proc. AFIPS Spring Joint Computer Conference, **21**, 113 (1962).

[2] L. M. HAIBT, *A Program to Draw Multilevel Flow Charts*, IBM Research Report RC-89 (April 1, 1959).

[3] I. E. SUTHERLAND, *SKETCHPAD: A Man-Machine Graphical Communications System*, Technical Report No. 296, Lincoln Laboratory, M. I. T. (Jan. 1963).

[4] J. T. GILMORE, *TX-0 Direct Input Utility System*, Memo 6M-5097, Division 6, Lincoln Laboratory, M. I. T. (April 10, 1957).

[5] C. WOODWARD, *UT-3: A Direct Input Routine for TX-0*, Memo M-5001-1, Appendix III, TX-0 Computer, Department of Electrical Engineering, M. I. T. (July 25, 1958).

[6] T. G. STOCKHAM and J. B. DENNIS, *FLIT—Flexowriter Interrogation Tape: A Symbolic Utility Program for TX-0*, Memo 5001-23, TX-0 Computer, Department of Electrical Engineering, M. I. T. (July 25, 1960).

[7] A. KOTOK and R. SAUNDERS, *DDT—DEC Debugging Tape*, Memo PDP-4-1, PDP-1 Computer, Department of Electrical Engineering, M. I. T. (Feb. 15, 1962).

[8] R. H. CAMPBELL and T. N. HASTINGS, *FAPDBG—A Symbolic Debugging Aid,* Memo CC-216, Computation Center, M. I. T. (August 10, 1963).
[9] R. FABRY, *MADBUG: A Mad Debugging System,* Memo CC-247, Computation Center, M. I. T. (Nov. 1964).
[10] J. H. SALTZER, *TYPSET and RUNOFF: Memorandum Editor and Type-out Commands,* Memo CC-244-2, Computation Center, M. I. T. (Jan. 11, 1965).
[11] R. C. DALEY and C. GARMAN, *ED: A Context Editor for Card Image Files,* Memo CC-245, Computation Center, M. I. T. (Nov. 20, 1964).

DISCUSSION

O. M. ABU-GHEIDA: You mentioned that SKETCHPAD was used to construct the flow-charts. Was this done by just utilizing the drawing ability of SKETCHPAD, or did you have to add blocks for the flow lines and various parts of the flow-chart?

T. G. STOCKHAM, JR.: Certain blocks had to be added, but, basically, just the drawing ability of SKETCHPAD was used. SKETCHPAD has a lot more capabilities than those implied here, but in this case it was just used as a drawing device.

G. T. THOMPSON: I wonder if you'd planned from the flow-chart to go directly into some compiling language.

T. G. STOCKHAM, JR.: What you are suggesting is that you could take a flow-chart, go into, let's say, FORTRAN, and then go through the FORTRAN compilation. The graphical compilation efforts of the Lincoln Laboratory are characterized by an entirely different approach in which the flow-chart drawing *is* the language, not a vehicle for an already-existing language.

J. C. R. LICKLIDER: You didn't say much about one principle and one subprinciple, and I'd like to ask you about them. The principle is that of having the computer examine stated expectations, written out ahead of time by the programmer, and then, when these expectations aren't made good by the operation of the program, go into a diagnostic mode. The subprinciple is subject to your statement about traps. In my opinion, the times at which subprograms are "called" and control is returned from subprograms to their "callers" are very important phases in the operation of a computer program, and they are therefore strategic—they are also convenient times to trap. Do you have any experience with what can be done by shifting into an interpretive mode at just those times?

T. G. STOCKHAM, JR.: I don't have any such experience, but I would agree that it is an important principle.

J. C. R. LICKLIDER: What about the expectation?

T. G. STOCKHAM, JR.: The concept of expectation fulfillment had not occurred directly to me, except in the sense that you could put an unconditional breakpoint at one place, some conditional breakpoints logically

before it, and then achieve more or less the goal that you are describing. We recognize that an investigation of sophisticated methods of program interruption for debugging purposes is one of the most important tasks facing us.

F. W. BLAIR: In the list processing that you are doing, are you by any chance using LISP?

T. G. STOCKHAM, JR.: No. This brings me to an important factor. The type of topology and abstract relationships to be kept track of in this particular form of work can't be conveniently modeled with a simple, one-way list. One needs to have a threaded list, a two-way list, and, what is more important than that, one needs to have a fast path from any other place in a list to the top of the list. In a two-way list that is closed, there is no bottom or top, but there is a key member. If one wants to form trees and go backward through the forks, one has to have a way of getting from any place on a branch to the next joint.

F. W. BLAIR: Two one-way lists would do.

T. G. STOCKHAM, JR.: Two one-way lists will do, but you will have to go arithmetically from the midpoint to any key member. You don't want to do that. You want to go in a fixed amount of time from anywhere in the list to the head of it. That requires an additional pointer.

R. R. FENICHEL: I have two questions. First of all, given that you are not using LISP,[1] what language are you using?

T. G. STOCKHAM, JR.: You are asking me what SKETCHPAD uses, then?

R. R. FENICHEL: I didn't realize that all of your work was built on top of SKETCHPAD.

T. G. STOCKHAM, JR.: Yes, it is.

R. R. FENICHEL: You're not using AED then, to get the maximum flexibility?

T. G. STOCKHAM, JR.: No. SKETCHPAD uses a list-structure with the characteristics I just described. Similar list-structures can be implemented via the AED-0 compiler at Project MAC. This compiler is an ALGOL implementation, but it also supplies some features which enable the user to construct any kind of list. There is also a language being completed at the Lincoln Laboratory called "CORAL." This language manipulates an improved version of the original SKETCHPAD list-structure by means of a very concise and neat set of macro-instructions.

R. R. FENICHEL: Secondly, I should like to speak on Dr. Licklider's question about a possible interpretive system which could run at full machine speed until the junction between independent routines. It is almost correct to say that the IBM 7090 LISP interpreter can be run in exactly this mode. There are a few details of non-correspondence, but I don't imagine that they would be of general interest here.

T. G. STOCKHAM, JR.: Let me turn to a new thought. An important factor seldom taken into account is the need for special debugging hardware. Computers being designed now should be provided with a mechanism that may be adjusted under program control to allow interruption due to a variety of program states. This mechanism should allow a processor to proceed at or very near full speed while looking for special conditions that have been specified by a person debugging a program. In computers designed for time-sharing, similar checks are made for violations which trespass upon memory security, and the like. The Lincoln Laboratory TX-2 computer has such debugging hardware that has been in operation for many years. Unfortunately some of its controls are through toggle switches, and thus they cannot be adjusted by a program. Until this situation is corrected in the near future, time-consuming interpretation is the only means for implementing sophisticated debugging mechanisms.

J. M. BENNETT: I am very interested in your original remarks about the use of brightening. About eighteen months ago, we moved from an old cathode-ray tube machine, which had this convenient brightening effect in its monitoring display as a by-product of the way in which its memory was accessed, to one which had no comparable display facilities. With the old machine, we got some advantage from the brightening in that it often showed where data and instructions were coming from.

T. G. STOCKHAM, JR.: You mention data. One can, if one wishes, perform a data flow analysis on a program. Dan Wilde at M. I. T. is doing this sort of thing, not with the intention of making any drawings, but just from the analysis point of view. One can display data flow-charts and hang one's data probes on them and one's breakpoints on control flow-charts.

DISCUSSION REFERENCE

[1] MCCARTHY, J. *LISP 1.5 programmer's manual.* Cambridge, Massachusetts: M. I. T. Press, 1962.

9
Control Language*

A. J. Perlis
Carnegie Institute of Technology

[ABSTRACT]

A language was described for carrying out simultaneously large-scale ALGOL computation, program editing, and on-line desk calculator computation. The intent is to specify a single programming structure for all of the communication required by a programmer to work at any or all of these three levels of computation.

* The manuscript of this paper was not available at the time this book was published.

SESSION IV

New Areas of Application

10
Computer Graphics and Innovative Engineering Design*

STEVEN A. COONS
Massachusetts Institute of Technology

It is rapidly becoming clear that graphical communication with a computer is of very great importance in man-machine interactive systems. At the beginning of an innovative engineering (or scientific) investigation, graphical modes of thought are natural; the engineer instinctively draws a sketch of a mechanism, or a diagram of a circuit, or possibly a flow diagram of a computational procedure, or a block diagram of transfer functions for a system. Indeed, graph theory itself is a study of graphs and their application to all of these abstract information structures.

The engineer's sketch serves as a mnemonic device that greatly assists him in fixing and focussing his ideas. It is superfluous to point out, for example, that an incidence matrix, while completely describing a graph, is a poor substitute when it comes to being an aid to human intuition and understanding. A similar reflection applies to tables of values that describe functional relationships; a graph is immediately clear, while numbers are not.

The early stages of innovative engineering activity in design are largely unstructured; there is no detailed algorithm that describes the heuristic, unpredictable process of creative engineering, or creative thought for that matter, for the same reason that there is no universal algorithm for constructing algorithms to solve problems, and it is certain that there never will be. This is not to say that certain intellectual procedures that men perform today by an exercise of art cannot in the future be formalized and subsequently mechanized, but it is to assert that there will always be an indefinitely extended hierarchy of intellectual procedures that

* The lecture originally presented at the symposium was entitled "Computer-Aided Design." This amplified version of a similar paper is used here at the suggestion of the author.

remain above the boundary of mechanizable tasks, no matter how far we succeed in expanding the boundary. It is at the interface between art and algorithm that the man-machine interactive computer system has meaning and potential. And one of the important modes of manipulation of ideas by mutual action of man and computer is the graphical mode.

As is no doubt well known, the actual physical implementation of computer graphics usually involves a cathode ray tube or 'scope, a light pen or equivalent device for drawing and manipulation of the graph, an associated keyboard, and possibly an array of push buttons and toggle switches for designation of certain frequently used subroutines and macro instructions, all of this peripheral equipment being tied to a computer. The state of the art is one of vigorous change, improvement and growth, and there are many human engineering problems yet to be resolved about the best, simplest, least expensive, and most convenient console configuration that produces ideal coupling between man and machine.

The Sketchpad System

The classic step toward graphical communication with a computer was, of course, Ivan Sutherland's SKETCHPAD program,[1] written for the MIT Lincoln Laboratory TX-2 computer, completed in 1962 and probably familiar to most computer people. Sutherland, far beyond merely modeling his graphics capabilities on traditional paper and pencil methods, introduced many transcendental notions into the system. SKETCHPAD is not a passive drawing system, an expensive and precise replacement for traditional devices and methods; it is instead a system that actively participates and assists the user.

The notion of applying a set of constraining relationships between elements of a graph, and the subsequent automatic relaxation of the geometry of the graph or drawing until these constraints are satisfied or until the discrepancies are at a minimum, makes it possible to perform many very sophisticated constructions with SKETCHPAD that would be quite difficult or at best tedious and confusing by conventional geometric or graphical processes.

For example, it is possible in principle to perform graphical field-mapping, the delineation of the potential field of ideal fluids (like electrical flow in conducting plate or magnetic fields, or heat flow, or water flow through earth). Briefly, this consists of sketching an array of flow curves within some flow boundary, and an orthogonal array of equi-potential curves, and modifying and resketching these curves until the resulting net consists everywhere of small squares.

[1] I. E. Sutherland, "Sketchpad, a Man-Machine Graphical Communication System," Lincoln Laboratory Technical Report #296, 30 January 1963.

In practice, this graphical procedure is extremely tedious and confusing; if the two families of curves are kept orthogonal, the small rectangles will not at first be geometrically similar. Some will be square, but others will be long and narrow, while still others will be short and wide. This violates the requirement, and further readjustment must be done until the square condition is met. With SKETCHPAD, the computer maintains the constraints on all the small quadrilaterals automatically, and eventually achieves a proper map. Of couse, such a procedure is not a very efficient way to solve such problems on a computer, but it is illustrative of the power of the notion of constraint satisfaction.

Compound Constraints

Again, in SKETCHPAD we see the germ of the idea of building compound constraints out of what might be called primitive or atomic constraints. Furthermore, such compound constraints are constructed graphically by a process that could be called "ostensive definition"; that is to say, the computer is shown how to do something for a special case, and then it is possible to copy the picture of the definition of the procedure and apply it to other cases.

For instance, suppose we wish to make six lines parallel and also equal, by pairs. We set aside temporarily the drawing of the six lines, and start with a fresh "sheet of paper" on which we draw two lines. These will serve as dummy variables. We now call for the constraint that makes lines parallel. This can be caused to appear as an abstract symbol or ikon, on the 'scope, and consists of a small circle containing the letter P, with four radiating lines or tentacles.

We attach the ends of the tentacles to the four ends of the two lines. The computer has been instructed to apply the "make parallel" constraint to the lines. Next we call for the "make equal" constraint. This also appears as an ikon, and resembles the "make parallel" constraint. When we attach its tentacles to the four end-points of the lines, the computer has been instructed to perform a compound operation on the lines. In Sutherland's system, no action will be taken to satisfy these constraints until the operator specifically commands that it be done, by pushing an appropriate button.

We now store the ikon of the applied compound constraint, and recall the drawing of the three pairs of lines. We can now call for an instance (loosely, a copy) of the ikon of the compound constraint, and can "merge" or attach it to each of the two lines of each pair. In this operation, the computer replaces the dummy lines of the ikon with the actual line of the problem. When finally the constraint has been applied to all three pairs of lines, a push of the "satisfy constraints" button will cause the lines to become parallel.

SKETCHPAD deals entirely with geometry; although by artifice, it is possible to solve some problems that are not inherently geometric, there are many cases where it is either very awkward or even impossible to communicated meaning. William Sutherland (Ivan's brother) addressed himself to the broad problem of computer graphics, with the aim of making it possible to communicate abstract procedures of any kind to the computer, including not only the geometric ones of SKETCHPAD, but others that the user may in the future define for himself, either through the keyboard or by construction from a base of primitive operations already in graphical form.

In this program, a graph or diagram can be drawn composed of any constituent elements whatever; the meaning of these elements can then be defined, and the diagram can then be "activated." For instance, one can draw a diagram of the arithmetical procedure for extracting the square root of a number, and then can introduce a specific number at the input terminal of the procedure graph on the 'scope of the computer, and obtain the result at the output terminal of the diagram if one wishes, by causing the computer to halt after each step; in this case, the various elements of the graph blink as they perform an operation on information. This makes it quite easy to "debug" the graphical procedure diagram.

This graphical technique has already been used with a programmed electrical network simulator, so that a sketch of an electrical network is sufficient to provide machine code for the simulation computations.

Computer graphics differs from pencil and paper graphics in another extremely important aspect; it permits dynamic behavior of the graph. The moving parts of a mechanism can be shown in motion, and such motion adds immeasurably to the information content of the drawing as far as the observer is concerned. It is possible not only to design and delineate a device, but one can actually "make it work" and observe its behavior. In principle we might not only watch the moving geometry of a mechanism, but we might also observe the deformations of the parts of the device under the influence of varying inertial forces superimposed on the static loading.

Free-Form Design

The design and subsequent detailed description of the objects with doubly-curved free-form surfaces is a very important and fruitful field for implementation by computer graphics. Airplanes, ships, and automobiles are examples of such free-form or "sculptured" shapes, and traditional methods for the design and production of such shapes are extremely tedious and slow. Sculptured shapes also occur around us in the small as well as in the large; the hand-set on a telephone is such a shape, as is

the cream pitcher on the table, the differential housing on an automobile, and the walnut stock on a shotgun.

Small objects usually are produced by artisans who sculpt models or patterns or sink forming dies based upon drawings furnished by the designer. These drawings usually consist of a few definitive design curves that depict and define the important aspects of the shape the designer has in mind, and leave to the artisan the job of interpolating the surface that will contain and agree with the intent of these curves. Of course the artisan and the designer can be in frequent communication with one another, so that if the designer has failed to be explicit enough about his intentions, he can add information, or can correct a misinterpretation in case the artisan goes astray.

In the case of the large sculptured shapes like airplanes, ships, and automobiles, such a procedure is of course out of the question; the designer must in some way define the surface explicitly and completely to a sufficient degree of detail so that further interpolation is mechanical and hopefully unique. Traditionally this has been done in the shipbuilding industry by an extremely long and tedious graphical process called "lines fairing" or "lines lofting," in which the ship is essentially drawn full size in several views or projections, and represented by a large number of plane sections or contours, somewhat like a topographic contour map. In lines fairing, it is by no means a simple, straightforward process to draw the section contours at intervals along the hull and thus generate a smooth "fair" surface. Instead, it requires many weeks or even months of drawing, cutting trial longitudinal sections, and smoothing of these sections by readjustment of points along the curves before the shape is free of unwanted bumps and hollows.

In the aircraft industry these traditional shipbuilding techniques were employed up until about 20 years ago, but have since been replaced by certain mathematical techniques that remove all of the older graphical trial-and-error procedure. But even so, the delineation of an airplane fuselage is a fairly complicated and time-consuming operation.

The design or "styling" of an automobile body is done by still another technique. Usually the designer creates sketches of a proposed new automobile, the best sketch is selected, a full-size drawing is prepared in color, and then a full-size clay model is made, with scrupulous attention paid to surface quality and overall authenticity. After changes are made on the model and it is finally approved, it is measured by elaborate and tedious processes and converted into section contours on a full-size drawing or "body draft." The measurement of the model and subsequent conversion to contour curves inevitably introduce errors into the information, and this "noise" has to be smoothed out by a process resembling roughly the fairing procedures of the ship loft. Eventually the body draft furnishes

information to enable dies to be sunk to form the panels of the body. This entire process is one-dimensional and very long and articulated, and at every joint of the articulation the information is contaminated as though through a leaky pipe. It is remarkable that the finished automobile resembles as closely as it does the original intent of the designer.

The Sympathetic Patternmaker

In all of these instances of shape design and description, traditional methods are long, tedious, and subject to error. Computer graphics will, in the near future, permit the designer to create the shape of an automobile body, a ship's hull, an airplane fuselage, or a tobacco pipe with consummate ease. This computer will behave like a skilled, sympathetic, and experienced patternmaker, or like an incredibly fast loftsman, or like a super-sculptor, working from meager information furnished by the designer at the computer console. This potentiality has come about fairly recently through the development of very general and powerful methods for the design and description of the entire class of sculptured shapes with the aid of the computer.[2] It is now possible for the designer to draw a few salient design curves and to have the computer automatically fit a surface to these curves. If the first computer interpretation of the designer's intention is not satisfactory, the designer can modify the curves already drawn, or he can add new curves to make his description more explicit. The computer will then automatically modify the original surface to accommodate the new information. The computer can calculate sufficient information about the surface to enable the shape to be displayed to the designer on the 'scope in somewhat less than a second, and subsequent modification of the shape takes comparably short computation time. Once the shape has been defined, it can be displayed in perspective rotated into any position, and moved about in space in real time.

The same shape-descriptive algorithmic structure in the computer can be used not only to produce the graphical display, but can furnish much more detailed information to run a plotter to draw the shape to any desired scale on paper, or to carve a full-sized model of the shape in some soft plastic, like styrofoam, or to run a numerically-controlled machine to sink the dies for the final fabrication of the parts. At present it is fairly standard practice to calculate points on such surfaces to a precision of about 21 bits, or about seven decimal places. This is entirely adequate precision for virtually all engineering purposes. Of course the precision of the arithmetic could easily be increased.

[2] S. A. Coons, "Surfaces for Computer-Aided Design of Space Figures," internal Project MAC memorandum, MAC-M-255, July 21, 1965 (to be published later).

The shape-descriptive information also forms the data base for other geometric calculations. It is possible to cut plane sections through the object, to obtain projected areas, surface areas, volumes, and to calculate the curve of intersection of two arbitrary shapes. The same data base also can furnish input to special programs for fluid dynamics calculations, three-dimensional stress analysis, and other analytical processes that require shape description and environmental description as inputs.

There is also a class of information structures which might be called quasi-graphical, consisting of mixtures of abstract symbols like words and numbers, but arranged in some familiar array or pattern. Matrices are excellent examples; we can cite also another example so common to us that if might be overlooked: the positional notation of our decimal number system itself, in which the geometry of position of digits is a means of conveying meaning concerning magnitude. The conventional integral sign with the appended upper and lower limits, followed by the integrand, followed by the differential operator and the independent dummy variable, is still another example of a quasi-graphical structure in which the array is a device for conveying meaning to the eye. Much of text-book mathematical notation has this quasi-graphical character, and it is important to preserve this structure when we are using a computer.[3]

The keyboard with its one-dimensional input-output string and its rigidly limited set of characters is certainly not well adapted to such forms of communication; a large encrustation of circumlocutions and makeshift techniques has formed around computer languages in an attempt to make them compatible both with people and with machines. The reason that quasi-graphical arrays on the printed page are easy for people to understand is probably that they make it easy for the human information processing system to construct efficient data structure models of their content, and these data structures are easy to manipulate and to remember. It is possible that similarly the two-dimensional array on the computer console can lead to a more easily constructed information model within the computer; this implies an enhanced transparency at the interface between man and machine. The man can more easily look into the computer, and the computer can more easily look back. We often use the expression "problem-oriented languages." It might be appropriate to say instead "people-oriented languages" to emphasize the goal of making communication with the machine truly natural. A measure of the degree to which some form of communication approaches the ideal is the degree

[3] Klerer and May, "A User Oriented Programming Language," The Computer Journal, July 1965, British Computer Society, pp. 103–108.

to which it is understandable weeks or months after it is written, not statement by statement, but in the structure of meaning that it reveals or conceals. Graphical communication is inherently structural; it seems likely that a truly fundamental effort toward implementation of such a way of man-computer conversation is the most important step to be taken in computer technology.

GENERAL REFERENCES

L. G. ROBERTS, "Machine Perception of Three-Dimensional Solids," Lincoln Laboratory Technical Report #315, 22 May 1963.

T. E. JOHNSON, "Sketchpad III—3-D Graphical Communication with a Digital Computer," M.S. thesis, Mechanical Engineering Dept., M.I.T., June 1963.

W. R. SUTHERLAND, "The On-Line Graphical Specifications of Computer Procedures," Ph.D. dissertation, Electrical Engineering Dept., M.I.T., January 1966.

11
Operational Military Information Systems

J. H. Bryant
IBM *Corporation*

INTRODUCTION

This paper presents some results from a continuing review of operational information systems. The study is being conducted to develop guidelines for evaluating the performance capabilities of existing systems and to obtain information which will be of value in planning modifications to existing systems. It is expected that the study will also provide a better picture of the environment in which systems under development must operate at some future point in time. This, too, will help identify the "cultural" factors likely to be encountered in implementing new systems and will aid preparation of effective implementation techniques.

Because there is often some question about what is considered to be an operational system, let me state the simple definition that I have used: a system which is reported to be used for some productive purpose. Further, I have been interested primarily in systems which are dedicated or committed for a significant portion of their time to the support of decision processes at some level of management or command—that is, those which are used to provide information in support of planning or operations. Because these definitions are quite general, selection of systems that satisfy these conditions has been somewhat arbitrary. But, since I have been interested in general conditions and in trends, this has not been considered to be critical.

While there has been no intent to make the study more restrictive than just described, the availability of and access to certain categories of information have influenced, so far, the direction and scope of the effort. Readily available information about military command-and-control and intelligence systems (as well as some personal familiarity with them)

and the opportunity to contact people familiar with military systems, have resulted in the command/intelligence group being reviewed first. The present report is concerned only with operational military information systems. The study is incomplete and, even when complete, will by no means be exhaustive. It is believed, however, that the results will generally represent what might be called the "state of the practice."

To indicate the orientation given to this report, let me indicate more specifically the kinds of things that have been of primary interest (also those things that have not been of interest). They fall into two categories. The first is simply the overall system environment, including such subjects as development history and trends and current modes of operation. I have not been concerned with particular hardware or hardware configurations.

The second category includes user/system interaction and performance capability. I have not been concerned with programming systems or the working programs as such, but more with the way and extent to which the systems permit expression or solution of information-handling problems from the point of view of those who create or use the information. To be more explicit, I have arbitrarily grouped subjects into the following categories, which seem to be common to all management or military information systems:

1. Creation and maintenance of information structures and organizations
2. Maintenance of information sets
3. Selection and presentation of information

Within each of these categories, or subdivisions of them, the objective has been to summarize the method or technique and the extent of expression. This has been done by comparing systems against an open-ended checklist with explicit parameters noted whenever applicable or possible. The present report presents only a summary of results. After briefly describing development approaches and current system environments, I will describe and illustrate what might be typical of existing systems. An attempt will be made to point out those features which are exceptions or not common to most systems.

DEVELOPMENT BACKGROUND

One of the most common ways of describing a military information system is to discuss the way in which it was developed. This is often of interest because the development approach has affected and does affect the end product and, in many cases, the capability of the system to respond to changes in its environment. It therefore affects the degree to which a system might reflect performance requirements in that environment.

System development approaches have gone through a number of cycles, which have been summarized by Ruth Davis* as follows:

1. The "Hardware" Era: 1953–57.
2. The "Damn the User—Full Speed Ahead" Era: 1957–60.
3. The "Don't Make a Move Without Calling Everyone" Era: 1960–63.

Another name for the last period might be the era of the "evolutionary system." All of the systems which have been reviewed to date were originated and have been developed during this period. All have had a somewhat similar development history in that they have evolved over a number of years. They do seem to fall, however, into three groups representing the origin of somewhat different species.

Figure 1 represents a system developed on a single basic equipment configuration, within a single operational environment. The system has been carried through more than one iteration or cycle of refinement within that situation.

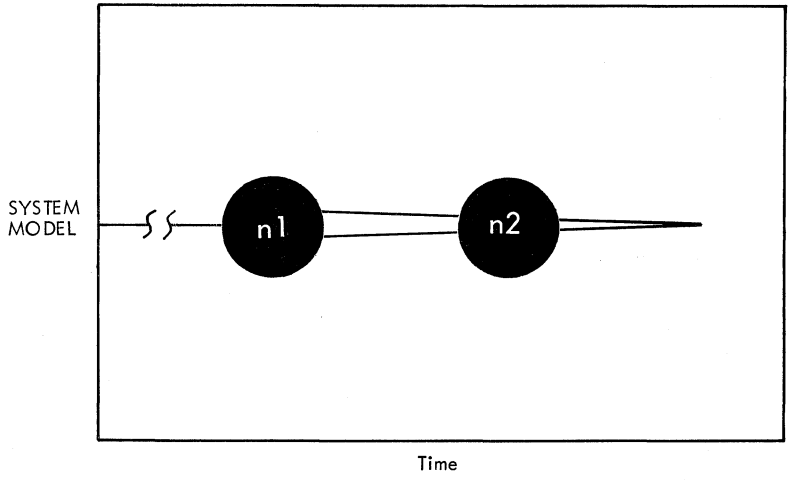


FIGURE 1. Typical system development process

Another type of development is depicted in Figure 2. It represents a system originally developed in one environment with one equipment configuration, but with a second-generation system planned for that

* Ruth M. Davis, Chapter 2, "Military Information Design Techniques," *Military Information Systems*, ed. E. Bennett *et al.* (New York: Praeger, 1964).

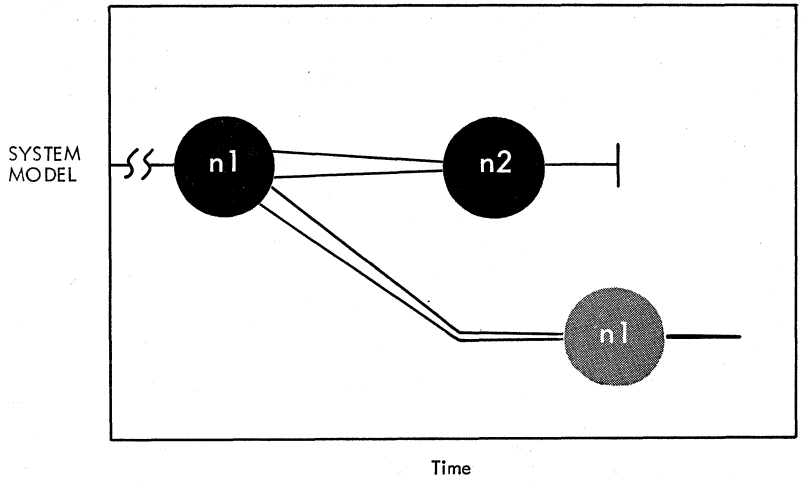


FIGURE 2. Typical system development process

environment with different equipment. It would not be surprising to find the pilot model "evolving," even though specifications for the more advanced model would be based on the first generation as shown here.

Figure 3 illustrates another group—an entire family of systems which have had interrelated courses of development. While this represents perhaps an uncommon situation, in the one instance encountered, it accounted

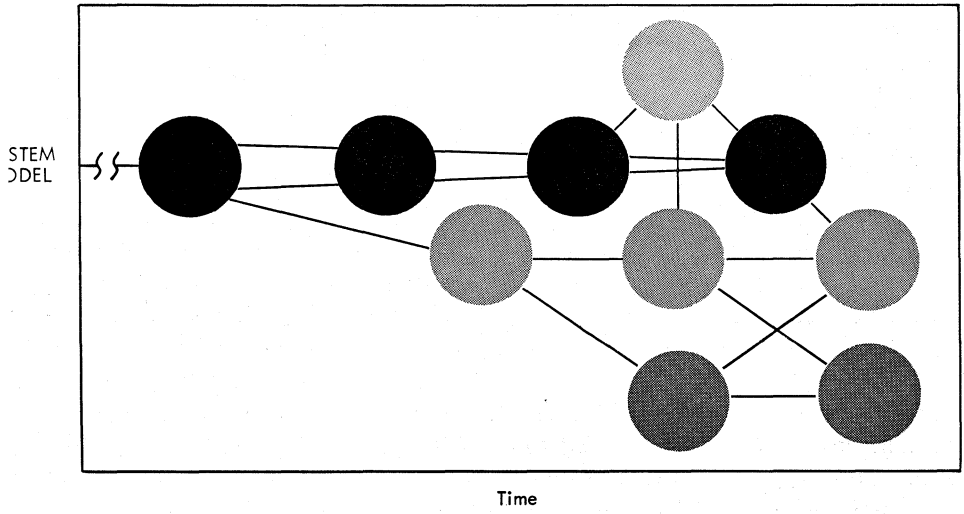


FIGURE 3. Typical system development process

for a large number (over ten) of operational or partially operational systems in different services and commands.

The systems reviewed so far have all been developed along one of these lines. All have been through one or more cycles of refinement based on successive experiences. They are, therefore, considered to represent some degree of responsiveness, because of this, to the requirements of the environments in which they have been developed and are being used. The extent to which and the order in which performance capabilities have been added are also indicative (holding technological capability constant) of the relative priority actually placed on classes of performance capabilities.

OPERATIONAL ENVIRONMENT

Before describing a composite military information system as it would currently look from the point of view of those factors mentioned earlier, let me comment about the operational environment in which the systems were developed and in which they operate and about some of their general characteristics. It is believed that this will help create an overall picture of the current state of technological practice.

Common to all of the systems reviewed is the concept of a centralized collection of information or data base accessible to all users within applicable security constraints. The data base is subdivided into logical information sets or files which serve some specialized purpose or support some specialized set of functions. The data base may be, but is most likely not, integrated so that logical relationship or interdependent information can be associated. Rather, information sets are typically created, maintained, and used independently by means of common facilities.

Most of the systems support the command in which they operate at a staff-planning level of organization. The typical user is an operations or intelligence analyst with little or no knowledge of or experience with data processing. A user may interact with the system directly but most likely will interact with an intermediary from the "system support organization."

The system support organization is most likely to be established, as a normal data processing organization, as a staff organization itself. The support staff will usually include, in addition to operating personnel, specialists capable of providing help in creating processable information sets, as well as in using the data base and, occasionally, in interpreting results.

Information sets are centrally cataloged by the system, and the catalogs are used as a means of interacting with the data. The catalogs are maintained at a detail level; that is, all elements of the set are specified in detail and provide dictionaries for translating user symbolic or textual terms into internal conventions, and vice versa. The catalogs may be

created in single or multiple physical counterparts, depending on the system. While the number varies from system to system, a typical system might maintain 6 to 20 major information sets which vary in complexity and extent from a few thousand to several million logical lines.

Most of the systems perform the basic functions of information storage, selective retrieval, and presentation in more or less general fashion. While they are by no means uniform, they have many characteristics in common.

Since most of the systems reviewed were started around or since 1960, most tend to be oriented toward serial storage devices. Direct access storage is used, however, in some capacity on most systems. Terminal devices, ordinarily typewriters, are used in some capacity on most systems. The number, typically small, provides the means of operator communication rather than serving as the normal means of user interaction. In some cases, terminals are available for analyst use, but these are mostly on an experimental basis or for training. Dynamic visual displays are rare, and even auxiliary graphic devices (such as plotters) are uncommon.

Most systems operate in a batch process mode, but it is not unusual to find blocks of time during prime shift reserved for open inquiry or analytical use. There are exceptions, but the usual situation is to find most (what may be called spontaneous) questions answered in a one- or two-day response period. Most inquiries are handled on a routine or standing basis as standard reports. These are prepared and periodically modified and processed—daily, weekly, or monthly.

In most of the systems the volume of incoming information is high, and, therefore, a large proportion of the time and effort is devoted to maintaining information. The rate of change in information sets does not appear to be unusual in comparison with other types of data processing installations but is, nevertheless, a significant factor in the situation. The rate of change seems higher in the way in which the data is used—that is, different information is selected, and it may be organized differently. New information sets are established not too frequently for a given installation.

This, then, is the overall characteristic of the systems and their environment. Now, let me describe how information problems can be expressed and solved with these systems.

PERFORMANCE FACTORS

The first category considered is the extent of information structures and organizations possible and the method used to establish or maintain them. This is considered to be an important factor, because it is assumed that there is no natural structure or organization which is inherent in all information and that structures are imposed principally for utilitarian

purposes—meaningfulness or efficiency. They may be meaningful in many forms but will be more or less efficient depending on the relationships established between elements and their use. In discussing this category, I am using the term "element" to mean the smallest useful information unit consisting of one or more bits. An information line consists of one or more elements which are always associated with each other.

Structures and Organizations

All of the military systems reviewed provide some means of explicitly expressing, establishing, and often maintaining structures and organizations. That is, structures are created and maintained explicitly at a detail level rather than existing as implicit controls in compiled programs. Most are oriented toward what are usually called formatted files, where a file or information set is composed of lines which, in turn, are composed of one or more elements. The lines may be of fixed length, but most systems provide some flexibility in structuring lines.

Figure 4 illustrates the concepts common to the typical information system in the current sample. As shown, it is composed of fixed-format line segments, some of which may occur a number of times in the line.

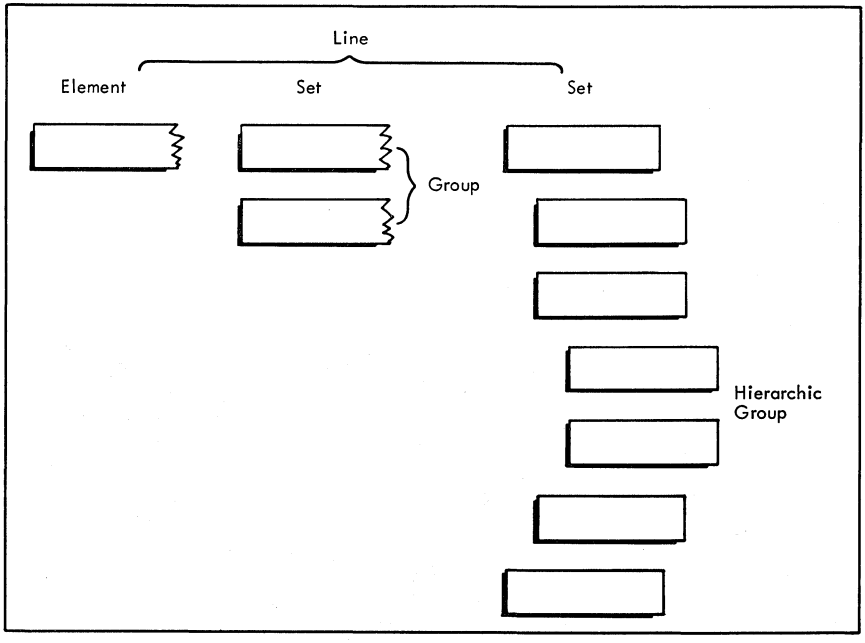


FIGURE 4. Variable-length, fixed format

All like segments compose a set or section. In some cases, but not usually, there may be multiple sets; and, further, a section may be made subordinate to some other segment to create some level of hierarchic relationships. In the typical system, data relationships must be maintained in contiguous physical space. It is an exception to permit more than one or two hierarchic levels. While the extent of a segment may vary, most systems do not provide for variable-length elements, although this is done in some cases.

File organizations are usually restricted to either an unordered collection of lines or a physically sorted collection. Indexing and chaining of lines or elements are permitted only in exceptional cases.

All of the systems provide facilities for defining such structures and organizations as might be desired, using some variety of card-creation forms or fixed-format statements. Shown in Figure 5 are typical job statements which might be used to create a data set in a typical system. The statements perform several functions related to the job and the elements of the job. The first entry in this illustration identifies the type of job which is being performed. The first few statements identify the data set to be constructed and provide control information for that set. The other statements name or identify controls for each element. In this particular example, the term FMOD is a convention which indicates that this is a file-creation job. The file is named and some priority assigned to the job. The numbers of fields and sets and groups considered to be

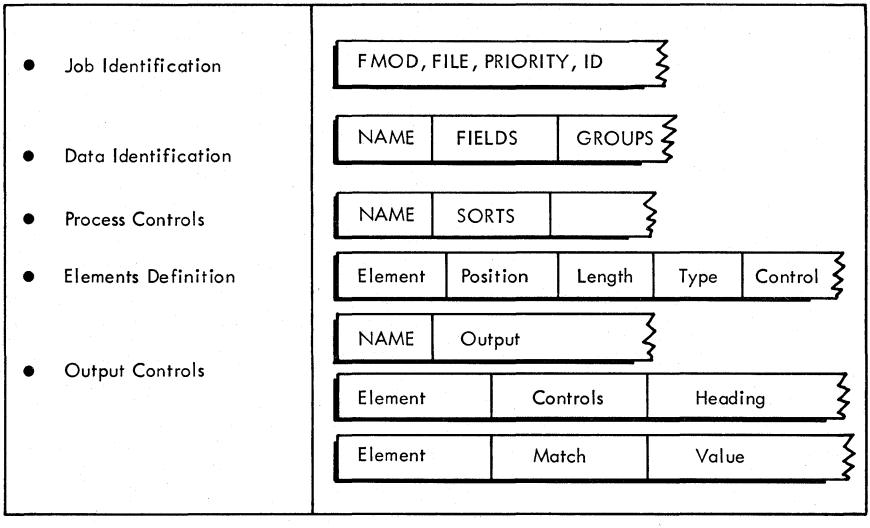


FIGURE 5. Expression of data relationship

a part of that file are indicated. Organization controls which pertain to the entire data set are also specified, such as might be used for determining the sort order of the file. Most of the systems reviewed provide for this type of function in one form or another, there usually being some limitation on the numbers of elements or at least the numbers of characters which could be used in creating a sort order. Specifications for composition and, in some cases, content checking are entered with the data definitions and are used to establish automatic legality checks as information is entered into a set. Finally, most of the systems provide some facility for specifying at data-definition time the headings and element values that will be used at presentation time. Essentially, the procedures followed in current systems are no different from what might be involved in defining or writing operating procedures for punched-card or manual files, other than that extra steps are required to create and name elements. They might be more systematic because of the fixed-format demands of the particular system. While the forms differ in detail between systems, they are generally similar, and the process of entering definitions into forms and/or processing such forms is roughly equivalent. It is significant, however, that once the definition is done, no further knowledge of it, as such, is required. It is used to translate references to the established element, group, set, and file by means of symbolic names.

The typical system provides some facility for altering established structures and relationships. This is done, in those systems where it is provided, by simply re-entering the data definitions in a fashion similar to that just described, with the actual data set being revised as an automatic function of the re-entry. In most cases, revision of the information set will require a specialized program.

MAINTENANCE OF INFORMATION

The next category to be discussed is the extent to which information sets themselves can be maintained and the methods that are used for expressing or accomplishing this function. Most of the systems distinguish between normal or routine and special maintenance, and they are performed in completely different manners. The typical system today will provide for the usual one-to-one maintenance situation: a single incoming line structure is transformed into a single system line structure. Provision is made for defining a single input and, as has been described, the system data set, as well as for making the transformation between the two. In some of the systems, provision is made to accommodate several incoming line structures and to convert them into a single data structure. This, however, is an exception. Only one of the systems offers a facility for receiving multiple incoming lines and transforming them into multiple storage lines, and this is not a generalized capability.

To summarize then, normal data maintenance—the adding of new lines to an established set—is set up at the time that the data definitions are prepared and is usually restricted to the case of a single incoming line being transformed into a single system data set. On an exceptional basis, additional facilities are provided for multiple inputs but none is provided for multiple inputs to be transformed into multiple files.

The definition typically provides for encoding from external language to a systems storage convention, in most cases checking the composition of incoming lines and in many cases checking for legal values. Most of the techniques involve straightforward checking of field length and character set, and looking up legal entries in a table. Programmed error correction is provided on an exceptional basis as "special" rather than as general cases.

A feature provided in the typical system, in one form or another, is the provision to add lines to an existing data set either on a routine batch basis, as I have just described, or in an on-line fashion from either cards or a terminal device. Figure 6 illustrates one of the ways in which this might be performed in one of the systems. As previously illustrated, the format for entering and the procedure used to enter such jobs are generally uniform, with three main elements contained in the job statement: (1) the job identification which addresses a specific data set, (2) the control statement which regulates or identifies the remaining information to follow in the job statement, and (3) the identification of an operation. In this case the operation is ADD, meaning add lines to an existing file, with the information elements to be added contained as a part of the statement. In this illustration there is no requirement to use a fixed format, even though the elements must appear in their proper relative order for the data set being addressed.

What might be considered special information maintenance operations are provided for in the typical system. Included in the category of special

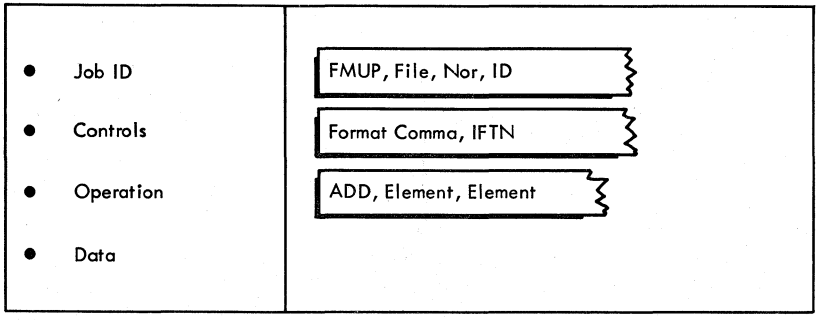


FIGURE 6. Expression of routine data maintenance

maintenance are those operations required to conditionally select and modify the contents of previously stored lines or to delete them altogether. In the typical system these operations are performed on a job basis as a result of detection of errors or the receipt of new or revised information. Conditional statements might be more or less involved, depending on the complexity of the problem. The typical system provides considerable flexibility in conditional expression. Since the conditional expressions used in special maintenance operations are more or less identical with those permissible in selective retrieval, more will be said about this later.

A typical job statement of this type is shown in Figure 7. The statement consists of three parts:

1. The job identification and control
2. A set of conditional expressions
3. The operations to be performed

The two types of maintenance commands typically permissible, however, are those of delete, as indicated here, and change. A change would be addressed to a specific element by name—such as, change the element A to some literal value. In some cases, the option is provided to change the contents of an element as a function of an algebraic expression.

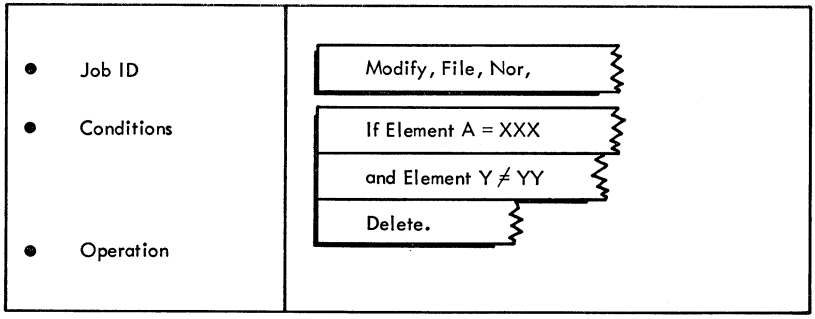


FIGURE 7. Expression of special maintenance operation

INFORMATION UTILIZATION

The subject of information utilization can be addressed with respect to three functional capabilities. The first topic concerns the facilities for expressing selection requirements or the retrieval aspect of data utilization. The second concerns the orginization of selected lines for presentation, and the third concerns the composition and presentation itself. These three functions are treated in all the systems.

In the typical system, the three functions are accomplished in a single job statement, as shown in Figure 8 for a simple case. As usual, the job statement addresses a data set. This is followed, as in the last example, by a given group of conditional statements which in combination express the selection requirements or retrieval requirements for that job. Following this are instructions for preparing the selected information for summary or presentation. In the case illustrated, this includes simply sorting on two elements in ascending order and a request for subtotals. No report instructions are provided, but rather a statement which indicates that the answer is to be returned on the typewriter rather than on a printer.

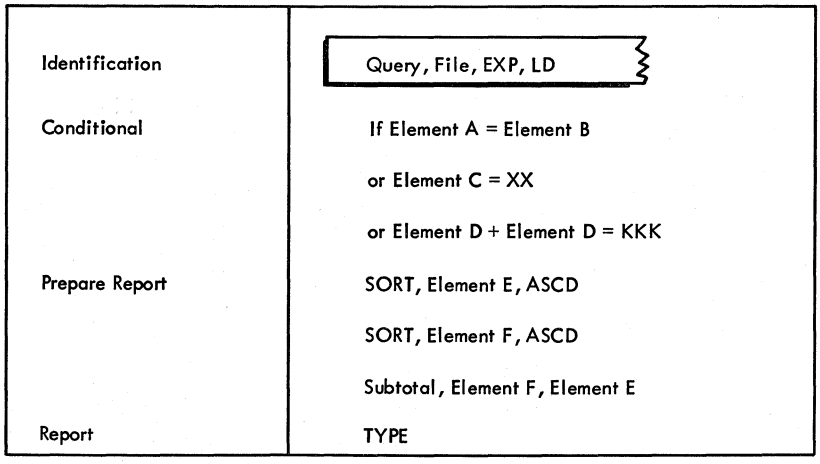


FIGURE 8. Expression of selecting and reporting

The typical system provides considerable flexibility in expressing conditional logic for information selection. Figure 9 shows the usual type of operators provided. As indicated earlier, elements are addressed by names combined by logical operators. In the typical system, the logical operators addressed to a particular element are related to a literal by connectors such as EQUAL, GREATER THAN, and LESS THAN. In some cases, usually in a very restricted form, provision is made for comparing one element with another element. In some cases, provision is made for comparing an element with an algebraically derived element. Most of the systems offer a number of specialized expressions (added at some time during the life of the system) to overcome the limitations that were built into the original programs. For example, in one system a logical operator, ALL, is used to indicate that the statements following will apply

Conditions	AND OR NOT Specials	= > < Specials	Literal Element Algebraic
Report Preparation	Sort Count Total Specials	Elements/Levels Elements/Controls Element Count	
Report Control	Select	Element	
	Position	Element	
	Title	Reports, Headers	
	Control	Formats	

FIGURE 9. Inquiry and report expressions

to all succeeding exclusive-OR conditions. As in another example, capability is provided to express the concept "within the boundaries of a convex geographic polygon." These special operators provide considerable extension to the logic capability of the system but compound the language.

In data preparation usually a small number of verbs or commands are provided. Those shown here are typical operations to be found in existing systems—sorting, counting, distributing, totaling, and so forth. In a typical case, a number of restrictions will apply to these commands, such as the amount or levels of sorting, distributing, or subtotaling. Obviously, some systems provide more verbs (processing commands) than other systems do, but those shown are typical.

In only one system is provision made for addressing multiple files in a single statement and for combining lines retrieved from each of the files. That is, new lines may be created which are composed of elements of both data sets, or a new data set may be created in which the lines are simply merged. In the same system, provision is also made for using the results of a retrieval as a basis for a secondary search. Computation that might be required to prepare or present information, above and beyond the types of general organizing and summarizing shown here, is provided on a specialized basis only. One system has an elaborate

set of special programs which display and compare existing status with plans, but this is an exception.

In preparation of information for presentation, the typical system provides two or more options. One option is for a normal report which simply presents information lines as they were stored with proper conversion to text and with headings as defined in the data set catalogs. A second option provided is to select only the elements which are of interest. This would be done by naming the elements in the last line of the job statement—Type, Element A, Element B, etc. In either of the above options, special headings and titles could be inserted. Full report generation is provided in some cases, but it is more typical to find systems which have stored special-purpose report programs which are called and executed when needed. In one or two cases, provision is made for graphing and plotting. These, however, are unusual features.

SUMMARY AND CONCLUSIONS

This report has summarized a collection of information relating to operational military information systems. The intent has been to depict the "state of the practice" as an aid to assessing proposed system performance capabilities and to planning implementation techniques. An attempt was made to describe the overall background and environment of existing systems, as well as the techniques and capabilities provided for user interaction or user problem solving. In planning new systems for these requirements, it is assumed that all these factors will have to be taken into account.

DISCUSSION

R. W. Taylor: What specifically have you found in this investigation to help you in future systems?

J. H. Bryant: I think the things I have been specifically looking for, most of which are apparent in a simple review of this type, are the limitations with respect to logic and expressive capability provided in existing systems. I haven't tried to summarize the limitations here. This has been simply an attempt to summarize the way they look today.

H. B. Goertzel: What capabilities do you find that the user wants that are not handled by this type of operation?

J. H. Bryant: In general, I think they fall into two categories. First, there is a need for additional data logic. I think most users that I have talked with would like to be able to express their data in a form that's more like the way they typically handle it, rather than having to force it into some limited structure along the lines described. Most of the systems reviewed are quite rigid in their requirements for data structure, and users want additional flexibility to express their data logic and data

relationships. The other category is directly related: they want an easy method of defining input to storage transformations which would permit them to accomplish data maintenance functions without having to write specialized programs. That kind of capability doesn't exist in the current systems.

A. VAZSONYI: Is there any estimate of the required response time? Can you say something about how fast do users want answers?

J. H. BRYANT: In talking with people in military systems, if you ask "What kind of response time do you want?" you get an answer like "Instantaneous!" But, I think more and more people are trying to express their response requirements in relation to their need to take an action. This is a more realistic approach than asking for a blanket two-second or five-second response time.

A. VAZSONYI: What are the indications that on-line systems are required?

J. H. BRYANT: Most of the people in these environments expect to have on-line devices of one form or another in those systems as well as in the advanced systems which they are planning. Most of them are going through a straightforward, evolutionary cycle, starting from typewriters and going on to more grandiose, dynamic devices.

H. W. BROUGH: On any of the systems that you have looked at, are the files of dictionary, for the elements or groups or sets, themselves searchable files? Are they used for interrogation to find out what the system capabilities are?

J. H. BRYANT: In most of these situations they are themselves created as any other data set and are, therefore, addressable by using part of the system. It is surprising that they haven't been used for the purpose you indicate since it is possible to do so.

It is also possible to do a good deal of the publication work associated with systems development; one could publish all the data catalogs and coding conventions directly from data dictionaries, but this typically has not been done.

12
Computer Applications in Biomedical Libraries

FREDERICK G. KILGOUR
Yale Medical Library

Three biomedical libraries have developed computer applications to mechanize procedures, and two of these libraries are working on computerized retrieval of bibliographic references. First to enter the field was the National Library of Medicine (NLM) with its widely known MEDLARS project which Scott Adams has decribed in the April 1965 issue of the *Bulletin of the Medical Library Association* [1]. The Washington University School of Medicine Library has computerized acquisitions, serials, and circulation procedures [2, 3], and the Columbia, Harvard, and Yale medical libraries have computerized the production of catalogue cards and monthly accession lists. The Columbia-Harvard-Yale group looks forward to establishing a real-time bibliographic information retrieval system employing the same machineable information used to produce catalogue cards and accession lists.

The principal product of the MEDLARS project is the *Index Medicus*, a monthly subject and author index of about 3,000 biomedical journals. The *Index Medicus* is an indispensable bibliographic tool for every biomedical library, for it makes available the articles in the journals to which each library subscribes. The MEDLARS project also produces recurring bibliographies, each of which is a "periodic listing of citations pertinent to a given field of medical science selected from editions to the computer store" [4]. Examples of recurring bibliographies are *Rheumatology Bibliography* and *Cerebral Vascular Bibliography*. The MEDLARS project also performs demand searches of the magnetic tapes on which are stored the article citations with their respective subject headings. Requests for demand searches must be sent in to the National Library of Medicine through the local medical library. The principal value of this type of retrieval lies in the ability to search complex coordinations

of subject headings. Early in 1965, NLM was receiving requests for demand searches at the rate of about 20 a day. Currently, NLM is computerizing its technical procedures for processing books and serials.

The Washington University School of Medicine Library first computerized its serial records and next its circulation records. Subsequently, the library has computerized its acquisition procedures and is working on the production of book-form catalogues. The Biomedical Library at the University of California at Los Angeles is also computerizing its serial records.

Other computer applications of importance are a union list of serials, *Selected List of Biomedical Serials in Metropolitan Detroit* [5], and a similar publication which the Medical Library Center of New York is preparing [6].

The Columbia-Harvard-Yale Medical Libraries Computerization Project is based on the alluring premise that it is possible to prepare catalogue cards from machineable information which can also be used for the preparation of accession lists, book-form catalogues, and special bibliographies. Moreover, this same machineable information can be accumulated until there is a sufficient amount on hand to activate a real-time bibliographic information retrieval system employing a central computer. In other words, Columbia, Harvard, and Yale are continuing the production of their card catalogues while accumulating information for real-time computer usage.

Briefly, computerized catalogue card production begins with the cataloguer writing cataloguing information on an $8\frac{1}{2}$-by-11-inch worksheet. This data is then keypunched with a punched card being produced for each line of text to be on the catalogue card; groups of punched cards for each worksheet or title are called decklets. These decklets are then processed on an IBM 1401 computer having a 4K core and two tape drives. Depending on the last program employed, either the computer produces a set of punched cards which drive an IBM 870 Document Writer to produce the catalogue cards, or the catalogue cards are produced directly on the IBM 1403 printer, using an upper- and lower-case print chain.

In general, there are three principles to be followed in the preparation of bibliographic data for computer processing. First, each category of information must be identified by an exclusive code; second, the computer must be able to detect when it passes from one category of data to another; and third, there should be non-printing flags that can identify portions of data within each category. When these three principles are followed, the data is, in effect, locked into the programming.

In its *Index Medicus*, the National Library of Medicine employs its Medical Subject Headings (MeSH), and the Columbia-Harvard-Yale group has adopted these subject headings for its real-time bibliographic informa-

tion retrieval operation. Thereby, it is possible to use NLM's indexing in machineable form in the Columbia-Harvard-Yale system. However, the Columbia, Harvard, and Yale medical libraries use subject heading systems differing from MeSH for their card catalogues. Columbia uses Library of Congress headings, as does Yale, while Harvard uses an earlier version of MeSH. In the cataloguing process, subject headings to be used in the card catalogue are placed on the body of the card—as is the practice in most American libraries—while the MeSH subject headings are written on the worksheet below the bottom of the card. Formerly, the Yale Medical Library was using 1.65 subject headings per title in its card catalogue, but more recently it has been employing 10.4 MeSH headings for computer retrieval [7]. This increased depth of subject indexing of books should improve the retrieval process. Each decklet of punched cards contains the MeSH headings as well as the data for catalogue card production.

There are five main programs which process decklets. The first program writes the data to go on the catalogue card onto magnetic tape. It also sets up a sort control for each entry under which a card is to appear in the catalogue. In addition, it establishes the address where that entry can be located in the data. For instance, the entry may be a subject heading at the bottom of the card or a short title within the title category on the card. Finally, this first program sets up various error messages for certain types of incorrect data submitted to the program.

Perhaps the most difficult programming area is that of establishing the sort control under which cards will be alphabeted and, ultimately, entries alphabeted in a book-form catalogue. It is the book-form catalogue for which the alphabeting must be particularly precise, for the alphabeting of cards is only for the purpose of arranging them for manual filing. Characters having diacritical marks must be altered in some languages for filing and not in others. If a diaeresis occurs over an "a," "o," or "u" in German, this character is set up as "AE," "OE," or "UE" in the sort control. However, English characters with a diaeresis, such as the second "o" in "coöperate," are not so altered in the sort control. Similarly, "M' " and "Mc" are converted to "MAC." In setting up the sort control for titles or short titles, it is necessary to remove an article if it is the first word. Once again, this article removal is accomplished according to the language. Otherwise, the English first-person pronoun would be dropped along with the Italian article "I." However, some configurations of abbreviations and all numbers must be converted to achieve correct alphabetization. There is no way to program a computer so that it can recognize whether "St." should be alphabeted under "Saint" or "Street," or whether "2" should be filed as "two" or "zwei." When these impossibilities occur in the area on which sorting is to be done, a special flag indicates to the

computer that it will find the exact sort control at the bottom of the card. Here the cataloguer spells out the sort control for the computer.

The second program has a control card which determines the number of packs of catalogue cards to be produced and the types of catalogue cards to be in each pack. This second program then explodes the tape record produced by the first program into the total number of tape records equivalent to the number of catalogue cards needed. The program also sets up the pack control number for each card record.

The third program is a modified IBM package program "TTSORT"—two tape sort. There being no alpha-numeric sort for a 4K 1401 with but two tapes, it is only possible to sort the cards into their various packs, but they are not alphabeted within each pack. A larger configuration having four tape drives could achieve the alphabetical sort.

The fourth program then sets up tape images of each card in its final format and produces certain codes for the operation of the fifth program, which may be a program for producing punched cards for the 870 Document Writer or one for writing out the cards on card stock directly on the 1403 printer. In the latter case, cards can be printed, depending on the setting of a sense switch, either one at a time or two at a time, side by side. Of course the cards printed side by side are different cards.

If catalogue cards are to be written on an 870 Document Writer, computer-produced punched cards are used. An 870 Document Writer consists of a keypunch connected to an electric typewriter. The punched cards are placed in the feed hopper of the keypunch, and a plugboard for catalogue card production is placed in the machine. Starting the 870 Document Writer causes the cards to be fed into the keypunch, and as these cards pass the reading head of the keypunch, they activate the typewriter, which writes cards on continuous-feed card stock. The 870 Document Writer has one advantage—albeit a small one—over the 1403, since it can type in red.

The cost of 1403-produced cards appears to be about the same as those written on the 870 Document Writer, although there has not yet been sufficient experience with 1403 production to be able to make a firm statement. However, cards produced on the 870 Document Writer probably cost less than those generated by traditional techniques. At the present time, with 1401 time being charged at the rate of $30 an hour, cards are being produced for about 9 cents apiece.

Once a month at Columbia, Harvard, and Yale, the decklets produced during the month are put through a program which prepares tape records for an accession list. These accession lists do not consist of lists of cards but rather of entries which vary from 70 characters a line at Columbia to over 100 in the Yale list. The first subject heading is placed first and the call number last, rather than first as it is on a catalogue card. The

tape records are then sorted by subject and printed out either for multilith or photo-offset reproduction. Each month about an hour of personnel time and minutes of computer time are used to produce the copy for these accession lists. Formerly at Yale it required the time of one person for one week to set up the list. Computer charges have been about $20 for each list.

A working copy of a book-form catalogue was produced in the autumn of 1964 from the Yale data containing some 1,400 entries together with their MeSH headings. Once again, the lines for each heading were extended to over 120 characters, demonstrating the flexibility in handling the original data. Ultimately, the Columbia-Harvard-Yale project will produce book-form author, title, and subject catalogues for dissemination that will contain the listing of books in each library with imprint date of 1960 and later.

The Harvard Medical Library has been working on a technique for producing a KWIC index from decklets for the publications of congresses, symposia, and the like. It is particularly difficult for reference people in medical libraries to identify such publications when a user asks for one, but it is hoped that with a KWIC index (whereby the work will be indexed under each word in the name of the congress and the title, as well as the place and the date) it will be possible to identify rapidly the volume sought.

Librarians at Columbia, Harvard, and Yale medical libraries have been convinced that the primary process with which to begin mechanization is the cataloguing procedure. It is also fundamental to computerized bibliographic information retrieval. However, once the mechanized cataloguing has been achieved, it will then be possible to move in the direction of a total system by computerizing such activities as acquisitions procedures, serials procedures, and circulation records. Indeed, the group has already begun to consider and to test the mechanization of acquisitions procedures.

It is hoped that the real-time bibliographic information retrieval system will be activated toward the end of 1966. The principal goal of this system will be to effect an increased speed and completeness with which the library user is supplied with cataloguing or bibliographic information. The sponsors of the Columbia-Harvard-Yale project feel that it has the potential for producing the first major step in supplying cataloguing and indexing information since the introduction of the card catalogue in the last quarter of the nineteenth century and since the earlier nineteenth-century introduction of the abstract and index journal.

There are three categories of bibliographic information retrieval currently in existence, and the qualities which differentiate among the categories are the times involved in effecting retrieval. A library-type service is fast, and it is now possible often to obtain needed references

in a matter of minutes. However, sometimes if an extensive literature search is involved, these minutes may extend into days. Such an extensive bibliographic search is typified by the demand searches of the MEDLARS project, where the writing of instructions for the program and the sequential searching of tapes can be most efficiently performed over a period of a day or two. Finally, there is the type of search which intends to point at specific data within a document, and here the search may extend into a week or two. It is the first category of retrieval with which the Columbia-Harvard-Yale project is concerned, and the system designed by IBM for the project would have an elapsed time of but $4\frac{1}{2}$ seconds between the completion of an inquiry at a typewriter terminal and the beginning of the return of the answer to that terminal if the coordinate search were to be on four subject headings.

The real-time system will consist of a computer configuration with remote terminals. The computer will possibly be located in New Haven and will have associated with it a high-speed printer printing in upper and lower case, a card reader/punch, random access storage, and a multiplexer through which the terminals will communicate with the computer. The terminals will consist of a typewriter and a card reader so that the data from cards can be read into the central system from a remote terminal. Requests into the system will be entered through the typewriter, and lists of references will be typed back on the typewriter.

If a user were interested in obtaining references on the use of computers for information retrieval in science, he would look up "Computers," "Information retrieval," and "Science" in a list of MeSH headings. Here he would obtain a code number for each heading. He would then type into the typewriter "SS" (subject search) followed immediately by the first code, which might be "09632." Next he would type a pound sign (#) to indicate the end of that code. Then would come the second code followed by a pound sign, then the third code followed by a pound sign, and then another signal to indicate the end of the inquiry. The typewriter would promptly type back three subject headings so that the user could determine whether or not he entered the right codes. At this juncture, he could enter parameters limiting the search to the holdings of one library, to references in but one or more languages, and to titles published in certain years. If he did not wish to enter any of these parameters, he would then type "go," and the computer would start its search and begin to type back the references in less than four seconds. Those references which were books would also have the call number of the book. The number of references typed out will have an arbitrary limit, such as 15. If more than 15 references are turned up in a search, the first 15 will be typed out and the rest set aside for special call from the terminal, perhaps during a slow period.

Anticipated achievements of the real-time system include greatly increased speed and completeness in supplying references, as mentioned above. Moreover, it will be possible to search the catalogues of three libraries simultaneously. There will be, and already has been, an increased depth of subject indexing of books, and the computer can be taught always to search "See also references."

Book-form catalogues produced by the project will be catalogues of the whole network rather than of just one library. Prior to the introduction of the card catalogue in the nineteenth century, printed book catalogues were popular, and these were useful in libraries other than those which generated them. With the advent of the card catalogue, free flow of bibliographic information among libraries was stopped, for the card catalogue exists usually in but one copy [8]. But with printed book catalogues it will be possible to make the listings of library contents much more widely available. The library at Florida Atlantic University has already taken a lead in the dissemination of printed book catalogues and has been closely followed by the Toronto University Library.

In general, it is anticipated that computers will continue to be used in biomedical libraries to mechanize library processes and, similarly, to mechanize the retrieval of cataloguing or bibliographic information. This second mechanization is really computerization of the user's activity, not the library's processes, and should immensely increase the efficiency of the borrower's use of the library. It can be expected that, following these applications, it will then be possible to computerize searching through texts of documents, much as the user now does once he has a list of references, to select those containing information which the user needs. Activation of this procedure depends on development of techniques for mechanically reading books and journals into random access stores that are huge and inexpensive. Finally, it can be expected that data retrieval systems will be developed which will supply the user only with the data he needs to have and not with references to the documents from which he can extract data.

REFERENCES

[1] Adams, S. 1965. MEDLARS: performance, problems, possibilities. *Bulletin of the Medical Library Association*, 53:139–51.

[2] Pizer, I. H., D. R. Franz, and E. Brodman. 1963. Mechanization of library procedures in the medium-sized medical library: I. The serial record. *Bulletin of the Medical Library Association*, 51:313–38.

[3] Pizer, I. H., I. T. Anderson, and E. Brodman. 1964. Mechanization of library procedures in a medium-sized medical library: II. Circulation records. *Bulletin of the Medical Library Association*, 52:370–85.

[4] *Op. cit.* (ref. 1), p. 145.

[5] *Selected List of Biomedical Serials in Metropolitan Detroit*, 2d ed. 1964. Detroit: Library, School of Medicine, Wayne State University.

[6] FELTER, J. W., and D. S. TJOENG. 1965. A computer system for a union catalog: theme and variations. *Bulletin of the Medical Library Association*, 53:163–77.
[7] KILGOUR, F. G. 1965. Mechanization of cataloguing procedures. *Bulletin of the Medical Library Association*, 53:152–62.
[8] KILGOUR, F. G. 1965. Research libraries in information networks. *Proceedings of the Second Annual National Colloquium on Information Retrieval*. Washington, D. C.: Spartan Books, 147–54.

DISCUSSION

H. W. BROUGH: In a very similar environment, we have remarkably similar systems. I wonder if you have encountered two things we have found need for. One is the use of pseudonyms in subject headings; the other is the use of generic headings—e.g., "Automatic Data Processing" as a code automatically implies "Computers," this relationship being built into the coding rather than into the assignment of subject headings.

F. G. KILGOUR: To answer your comment on generic headings, we are using, as far as the real-time system is concerned, the MeSH subject headings of the National Library of Medicine. This is a pre-coordinated list of headings with "See" references and "See also" references. As for the use of pseudonyms, we run into two problems. First, we have to construct an additional list of subject headings which are the names of people and places. The second problem arises when there isn't a topical subject heading, which we need, in the MeSH list, so we invent one and suggest it to the National Library of Medicine. If it doesn't accept our suggestion but adopts another heading, no major problem arises. If we used "Infantile paralysis" and the National Library of Medicine said: "Wait a minute! Back in the 1930s we found out that more than infants had the disease; 'Poliomyelitis' is what to use," then we would change to "Poliomyelitis," but we wouldn't have to make a major alteration in the real-time random access file. We would just put a pointer at the end of the "Poliomyelitis" file to go to "Infantile paralysis."

M. TUROFF: I infer that you are establishing your own identification number for each item.

F. G. KILGOUR: Correct.

M. TUROFF: Are you attempting to use it for books? You aren't attempting to use Library of Congress catalogue card numbers?

F. G. KILGOUR: No, we are using our own.

M. TUROFF: In the real-time system, have you established how much random memory you are going to need for how many items?

F. G. KILGOUR: You are talking about the random access memory to be holding the two files?

M. TUROFF: Yes.

F. G. KILGOUR: We have worked this out. The National Library of Medicine has figures for the number of characters in its journal article

entries, and we have done a study with Peter Sprenkle of the New Haven IBM office to determine the number of characters on library 3-by-5 cards.* If we begin operations with seven years of book cataloguing and three or four years of journal indexing in the random access memory unit, we are going to need a capacity of over 30 million characters. One requirement of the system is that it be readily expandable. Of course in the future, we could publish in permanent printed form and erase from the memory unit, but I think that with the development of hardware this procedure will not be necessary. In fact, it isn't necessary now if you can afford huge devices.

V. O. McBRIEN: In mathematical circles, we have been having great difficulty in the past 15 years in the cataloguing of books. Several months ago this matter again became an issue in the *American Mathematical Society Notices.* In a letter to the editor a set of ten contemporary books was listed with the question: "Where would you catalogue these books?" One of the big problems is that of terminology. For example, a mathematician mentions orbit theory, and it doesn't have a thing to do with the technical "orbits" of a chemist, physicist, or engineer. This makes the problems of cataloguing and retrieval of mathematical works extremely difficult. Do you think that there is any hope of getting computer assistance to make a wider spread of Library of Congress numbers so that, for example, books on topology might get a decent catalogue number?

F. G. KILGOUR: This is a problem which machines aren't going to solve in the immediate future; actually, you have to subject-head and classify for the user, not for the content of the book. In many libraries, a book is classified *in vacuo.* For instance, at the Yale Medical Library we get quite a few books on statistics which are really on business statistics, but they are for the use of biometricians. We don't classify them under business in the social sciences area of the classification scheme. Nobody would look for them there. They look for them with the rest of the works on statistics, although the Library of Congress quite properly puts volumes on business statistics in another place. All kinds of subject classification problems of this nature are generated because of the different uses to which books are put.

J. R. BRATHOVDE: We seem to be embarking on a new venture as far as the automation of libraries is concerned. The staff of the Library and Computer Center at the State University of New York at Binghamton has been doing some serious thinking about this problem, also.

I'd like to throw out for the consideration of this group a few comments on, and to have your comments on, the apparent bottleneck to which

* F. G. Kilgour and P. M. Sprenkle. "A Quantitative Study of Characters on Biomedical Catalogue Cards—A Preliminary Study," *American Documentation,* 14:202–206, July 1963.

reference has been made. Cataloguing is done by abstracters in a back room, as you mentioned, who are not professionals like ourselves, but three-, four-, or five-thousand-dollar-a-year cataloguers or abstracters who do not understand or appreciate much of the content of the material to be catalogued.

I wonder if the professional societies might help solve this dilemma by suggesting to their member authors a mechanism to abstract their own books or articles. Some form of source-abstracting is both necessary and timely because the libraries are on the threshold of seriously contemplating machine manipulation of acquisitions and cataloguing. Now, an author should know better than anyone else what is implied or expressed in his work. The mechanism would be for the author to list ten to twenty-five descriptors, and the publishers could merely print the descriptors near the preface or, better yet, could key-punch the descriptors and include the five to ten key-punched cards in a pocket attached to each book. This might add ten or fifteen cents or even fifty cents to the price of each book, but it would be cheaper than having each library staff, first of all, abstract the book and, secondly, perform the manual operation of key-punching. The advantages would be uniformity, a necessary factor in efficient information retrieval, and, most important, having in the abstract the concepts desired by the author.

I realize that this is a formidable task, that of soliciting cooperation among publishers, but a solution of the problem demands positive action taken now while we are still on this threshold of the information explosion.

F. G. KILGOUR: Librarians have been working on similar arrangements for a long time. One procedure is known as "cataloguing in source." It was in part begun back in the 1880s but couldn't be made to work then; it has been tried since without outstanding success. In some learned-society publications, like the *Federation Proceedings*, authors do submit subject headings for their papers; they are given a list of subject headings and select certain headings, which are submitted with their abstract that is to be published. The index to the publication is set up by machine using the author-assigned headings, but there are real troubles because, although the author knows what he is writing about and for whom he is writing it, he cannot anticipate all of the uses to which his work will be put. For instance, the author of a book on business statistics, as I have already mentioned, would undoubtedly assign subject headings in that field and could not anticipate that some library would acquire it for the use of biometricians.

The complexity of the procedures involved in your suggestion is alarming, and some publishers certainly will not cooperate. The important goal for the foreseeable future is to have cataloguers who have substantial knowledge of the information in the books they are cataloguing.

SESSION V

Man-Computer Interaction in the Laboratory

13
Computation and Control in Complex Experiments

W. F. Miller
Stanford University

I. INTRODUCTION

Most of the work presented in this symposium has been concerned with problems that arise in the development of systems that permit a man rapid access to computational results. This paper is concerned with problems that arise in the development of systems that are employed for data analysis and control in complex experiments. Such systems might be represented schematically by the block diagram in Figure 1. Such a diagram has the same general structure as a guidance system, and, indeed, it should, since its function is to guide an experiment. Such systems, of course, give rise to many of the same problems already discussed; but, on the other hand, they introduce some additional ones.

The motivation for on-line data analysis and control systems for experiments is very much the same as the motivation for rapid access to computational results. Both developments are motivated in part by the hypothesis that rapid feedback is essential to learning. It is clearly hoped, and by now it may already have happened, that a researcher sitting at his console will make discoveries by seeing his results fed back practically instantly. In the experimental systems, it is also expected that a researcher, by having his data quickly analyzed and by having intimate control over his experiment, can make discoveries and direct the course of the experiment in a more profitable way.

Let us look at the scope of the problem. As physics experiments probe deeper and deeper into the fundamental constituents of matter, the experiments become much more complex, and the interpretation of the data becomes more subtle. A single experiment involving a high-energy particle accelerator and a piece of detection apparatus, such as a spark chamber or a bubble chamber, may take several weeks or even months to set up and may run continuously for equal lengths of time. The accelerators

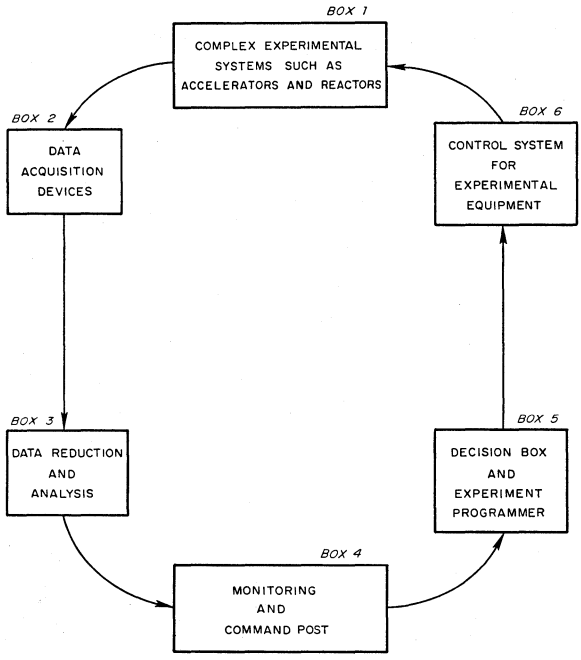


FIGURE 1. General schematic of integrated data analysis and control system for complex experiments

themselves are quite costly (several tens of millions of dollars), and they are expensive to keep in operation (several million dollars per year). The detection equipment is also quite expensive. A bubble chamber may cost several million dollars, and a spark chamber may cost several hundred thousand dollars. Clearly, there is an economic motive to provide data analysis systems that facilitate effective use of such expensive equipment. Rapid feedback of analyzed results is particularly important during the setting-up stage.

It may be worth pointing out that a typical spark chamber experiment conducted over a few weeks will generate 500,000 to 700,000 stereo views of events taking place in the spark chamber. The stereo views may be generated in pairs or triads, depending on the experiment. The manual and semi-automatic data analysis systems now in use require weeks and months to analyze this data. The Lawrence Radiation Laboratory at the University of California at Berkeley has had long experience on this problem and, by now, has good information on what it takes to analyze the data. Recent summaries [Reference 1] indicate that it takes about

seventy-five people to do the scanning, measuring, and maintaining of the equipment in order to process 250,000 events per year from the Berkeley 72-inch bubble chamber.

It is hardly necessary to pursue this motivational discussion further. It can be simply summarized by the statement that in the current era of experimental physics a disproportionate amount of time and effort is spent on the analysis of data and the control of the experiment. As a consequence, the experimentalist is removed farther and farther from intimate contact with the essential elements of his experiment. The goal of rapid data analysis and control systems in complex experiments is to bring the experimenter back into intimate contact with his experiments. Man-machine communication is required at a rather sophisticated level.

Over the last few years, I have been concerned with computer methods to achieve this goal. In particular, I have been concerned with the elements that would be contained in boxes 3, 4, and 5 of Figure 1.

II. CURRENT SYSTEMS

An example of a data analysis and control system of the kind depicted in Figure 1 is given in Figure 2. The system shown is a closed-loop analysis and control system involving a 3.0 MeV Van de Graaff Accelerator and its nucleonic measuring equipment. I shall not discuss this system in detail here, since it has been described elsewhere [Reference 2]. I should

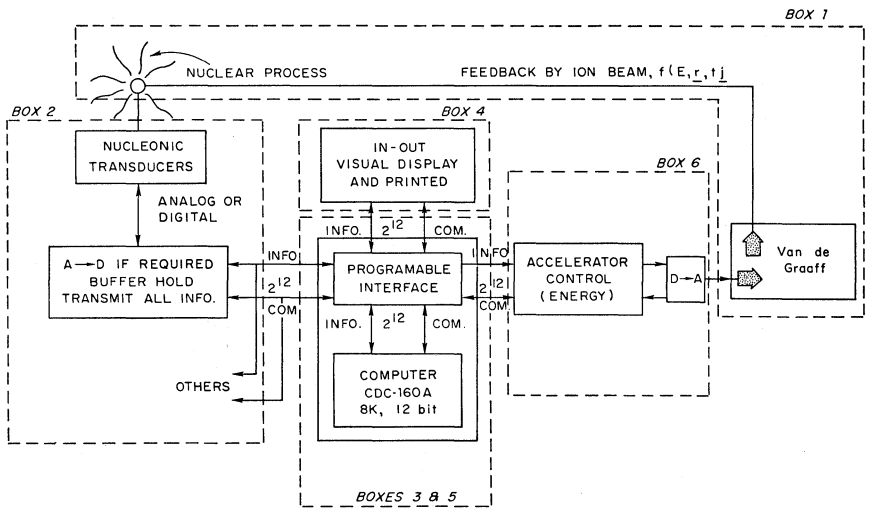


FIGURE 2. Closed-loop analysis and control system being utilized in low-energy nuclear physics experiments

like to point out, however, that this system has all the essential ingredients of the general system shown in Figure 1. It has, indeed, accomplished all the objectives that the physicists who were developing it expected.

The type of control that this system provides the physicist is shown in Figure 3. This flow chart represents the first experiment programmed on the system [Reference 3]. It provided the physicist a running statistical sequential analysis, configuration control over the sample, and energy control over the accelerator. The system has evolved very rapidly, and there are now several experiments programmed for the system [Reference 4].

This system is utilized in low-energy nuclear physics experiments of a type relatively well understood. As a consequence, the data analysis codes and the experiment programmer were not required to have the generality that would be required of such a system in high-energy particle physics. Nonetheless, some quite valuable experience was gained in programmed control of experiments.

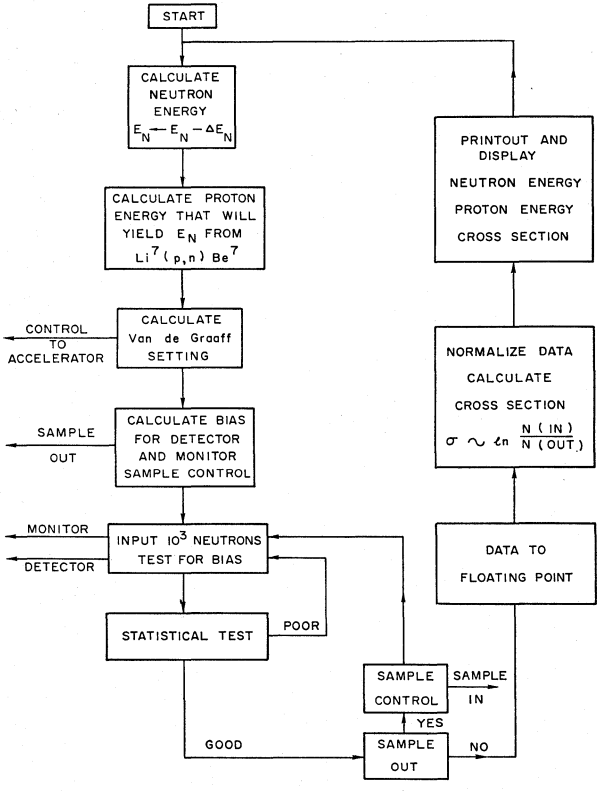


FIGURE 3. Schematic flow chart of first program run on the system depicted by Figure 2

Now, how do we stand in the very complex experiments such as we encounter in high-energy particle physics? We are certainly making progress in all areas indicated in Figure 1. At present, there are no high-energy particle accelerators operating under computer control. [By computer, here, I mean internally-stored program computer.] At two centers, however, computer controls are being developed. A digital control system, employing a CDC 924, has been developed for the Zero Gradient Synchrotron (ZGS) at the Argonne National Laboratory. It is expected that this system will soon be "hooked up" and that the ZGS will operate under its control. At the Stanford Linear Accelerator Center (SLAC) a computer control system is being developed for the beam switchyard. The beam switchyard will employ bending magnets to deflect the electron beam from the accelerator into different experimental areas.

The problems represented by box 3 of Figure 1 are in somewhat better shape. Several on-line data analyzers have been employed in low-energy nuclear physics [References 5, 6, 7, and 8], and one system has been developed in high-energy physics with great success [Reference 9]. At Stanford we are also developing a system similar to that described in Reference 9. This system is intended to analyze on line the data generated by the 20 BeV/C Magnetic Spectrometer.

We are also developing a computer system that will permit on-line analysis of graphic data of the kind we shall get from filmless spark chambers. There are two large general problems that have to be dealt with: (1) the control programs to permit processing of data from devices with high burst rates and (2) the data analysis techniques for handling the graphic data. Both of these areas are getting a great deal of attention and are important to the development of integrated systems. I shall give below a discussion of the type of programs developed to handle the data analysis part.

III. GRAPHIC DATA PROCESSING

A. General Description

In our work at Stanford, we are taking as a point of departure the Argonne work on automatic film data processing. That work was the collaborative effort of several people in the development of a film digitizer, programs for running the film digitizer on line to a large computer (CDC 3600), and the analysis programs for handling the digitized film data [Reference 2].

Although the Argonne work was done explicitly for film data, the programs maintained sufficient generality so that they can be used for any graphic data from spark chambers, no matter how the data is pre-

sented to the programs. Figure 4 shows the structure of these programs. The scanning and measuring program AROMA prepares, for the AIRWICK programs, the digitized information from the photographs. The AIRWICK programs identify the corresponding sparks in the two stereo views, calculate their positions in three space, and then link these sparks into tracks.

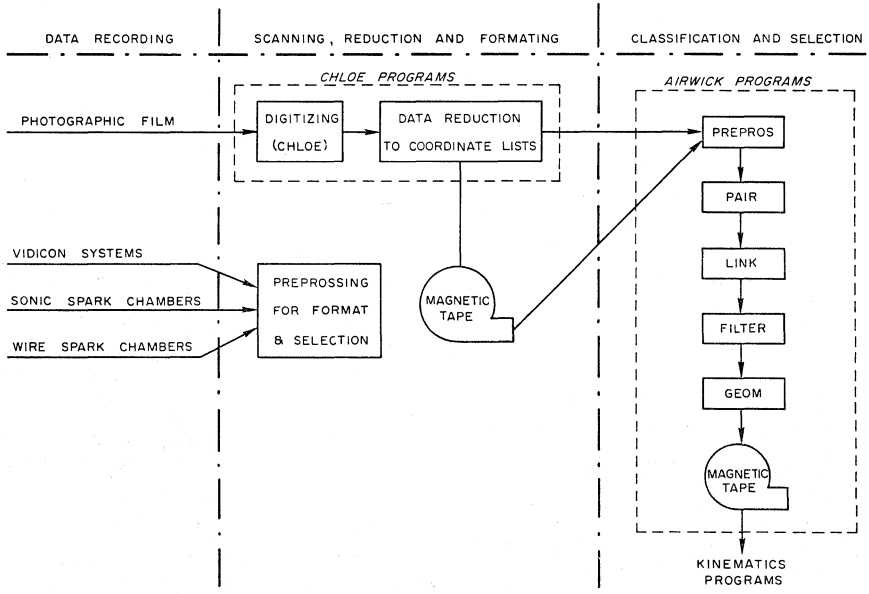


FIGURE 4. Schematic of the flow of data in the Argonne system

Figure 5 shows a stereo pair of photographs of an event taking place in a spark chamber, and Figure 6 shows the results of two stages of processing—after AROMA and after AIRWICK.

One of the most pleasing aspects of this work is that we can give a formal description of each step. Formally, the problem is presented as (a) the generation of a graph, the vertices of which are given by the three space coordinates of the sparks, and (b) the selection of the proper tree (or forest of trees for multiple events) to represent the event. The processes—physically described as (1) scanning, measuring, and image transformation, (2) pairing in the stereo views, and (3) linking into tracks—have their counterparts in the formal description. The first two of these

[*Text resumes on page 121*]

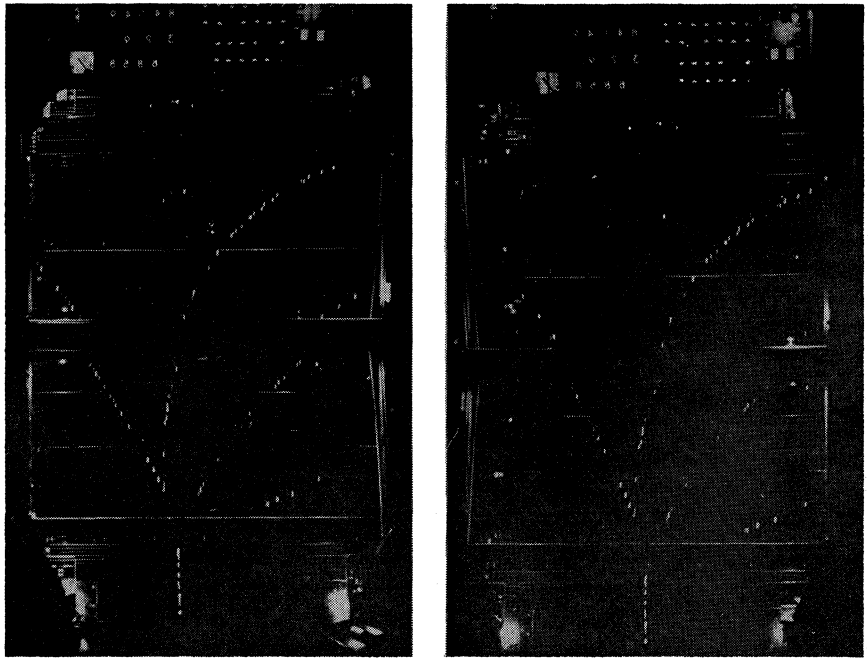
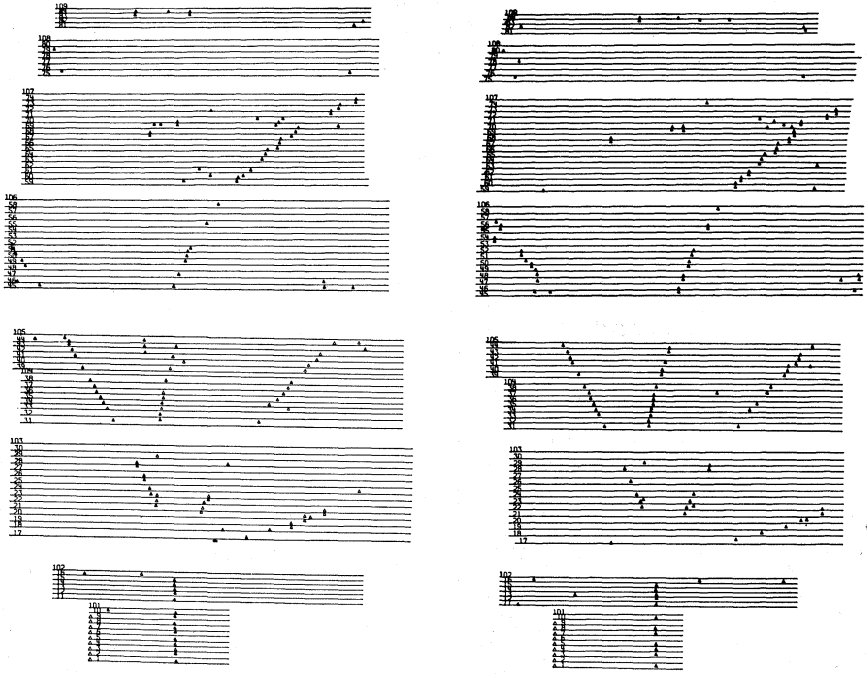
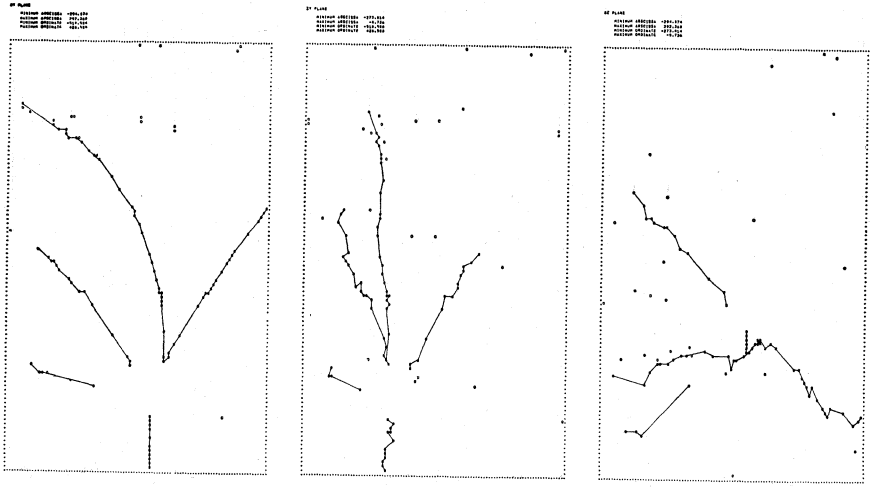


FIGURE 5. Photograph of spark chamber event in two views



(6a) Measurements from the same two photographs after processing by AROMA



(6b) The event after processing by AIRWICK, showing the reconstructed tracks. Random sparks not on the tracks have now been eliminated.

FIGURE 6. Output of AROMA and AIRWICK for event shown in Figure 5

processes (scanning, etc., and pairing, etc.) are concerned with the generation of the graph, and the third (linking, etc.) is concerned with the tree selection. Figure 7 shows the relationship between the physical processes and the formal description. The essential elements of the system are described in brief detail below.

Let me first give some data on the rates of manual systems and then those of our automatic system. Using the manual systems and skilled human operators, one can expect to scan and measure at the rate of about 20 to 30 events per hour. We are able to scan and measure at about an order of magnitude greater speed. We can process a stereo view of complex nature in about 20 seconds and simple ones in 10 to 12 seconds.

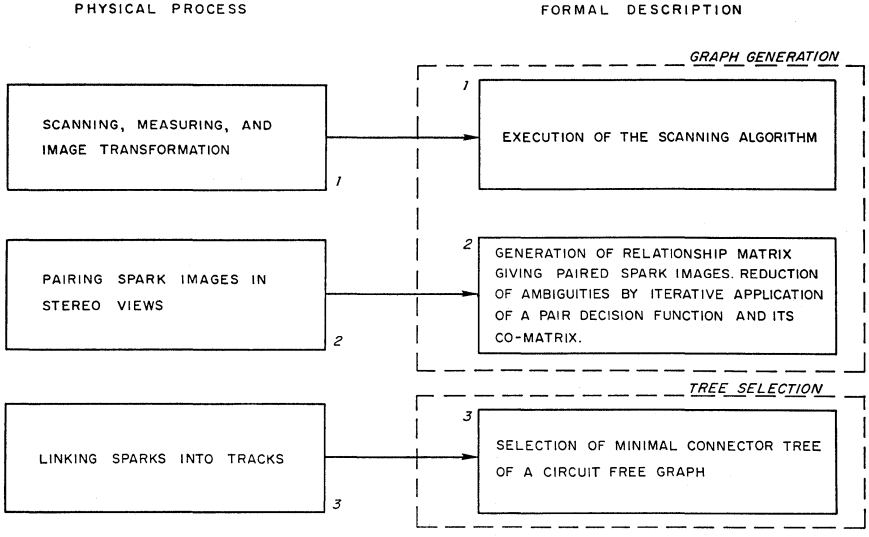


FIGURE 7. Schematic of the relationship of physical processes to formal description

B. Scanning, Measuring, and Image Transformation

The CHLOE film digitizer is described elsewhere [References 2, 10] and will be described only briefly here. CHLOE is a hardware system for digitizing data recorded on transparent 35-mm film. The hardware consists of (1) a controlling computer, (2) an optical scanner (CRT) operating under the control of the computer, and (3) a data link to a larger computer. A spot from the cathode-ray tube is projected on the film, and the transmitted light is measured by a photo-multiplier. From

this measurement a decision is made concerning the density of any rectangular portion of a 4,096 × 4,096 raster measuring 1.25 inches on a side. In practice, one scans only a small portion of this raster.

The scanning and measuring functions are carried out in the CHLOE LIBERATOR program which resides in the controlling computer (an Advanced Scientific Instruments 210) and in the AROMA program which resides in the CDC 3600. The principal feature of these codes is the CELL CONSTRUCTION ALGORITHM. The CELL CONSTRUCTION ALGORITHM may be described as follows.

Let us assume we are engaged in horizontal scans. The computer permits a left-to-right scan across a window defined on the raster by left-right limits and top-bottom limits. A point of interest is recorded when the scanner detects a change in intensity of transmitted light. As a consequence, in scanning across a spark, one gets first the left and then the right end of a line segment, the left end of which is the first point in the spark and the right end of which is the first point out of the spark (see Figure 8). If several sparks are encountered in one horizontal scan, one obtains ordered pairs of left-right coordinates of line segments, i.e.,

$$x_{1,\text{left}} < x_{1,\text{right}} < x_{2,\text{left}} < x_{2,\text{right}} \cdots$$

The CELL CONSTRUCTION ALGORITHM sorts these line segments into cells, eliminates some of the cells as clearly not being sparks, and calculates certain spark parameters such as area, centroid, and average width.

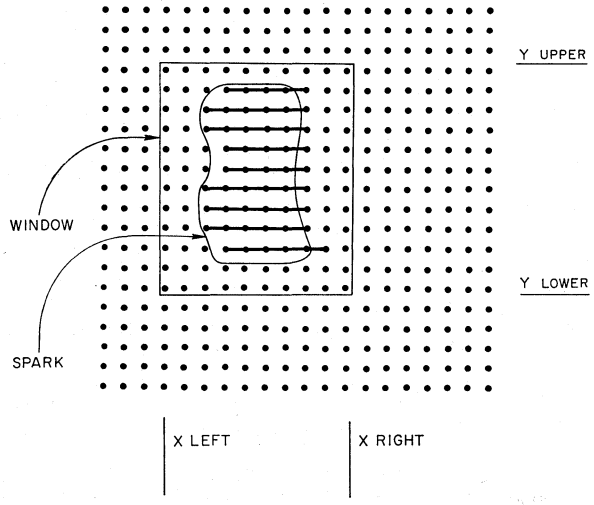


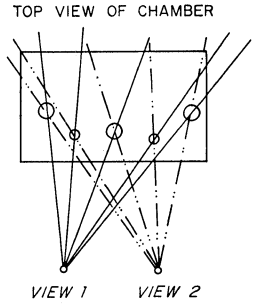
FIGURE 8. Line segments and cells generated by CELL CONSTRUCTION ALGORITHM

The AROMA program also provides information on the fiducial marks which are used to provide the orientation of the film coordinates to the spark chamber coordinates.

The next step is to identify the corresponding spark images in the different stereo views and to generate the three space coordinates of the sparks. Before this is done, the data is passed through a program (PREPOS) whose function is to remove optical distortions and to transform CHLOE film coordinates of spark centroids into real-space coordinates on the surface of the spark chamber.

C. Pairing in the Stereo Views

Since the separate gaps of a spark chamber are easily discernible, the pairing problem need only be concerned with pairing spark images in the same gap. The first step in the pairing is the generation of an n-dimensional relationship array—one dimension for each view. Figure 9 shows the two-view relationship matrix that would be generated for the five sparks generated in the spark chamber. The relationship matrix contains a 1 if it were geometrically possible for the rays to have originated in



RELATIONSHIP MATRIX

$j \rightarrow$ VIEW 2

VIEW 1 ↓

1	1	0	0	0
1	1	0	0	0
0	0	1	0	0
0	0	0	1	1
0	0	0	1	1

FIGURE 9. Schematic of top of spark chamber with five sparks and corresponding relationship matrix

the chamber and a 0 if it were not geometrically possible. The PAIRING ALGORITHM removes the ambiguities from the relationship matrix. The ambiguities are removed by application of a pair decision function

$$D(i, j) = M(i, j)[w_1F_1(i, j) + w_2F_2(i, j) + w_3F_3(i, j)] \tag{1}$$

and its co-matrix

$$A(i, j) = \sum_{n=1}^{v} D(n, j) + \sum_{m=1}^{s} D(i, m) - 2D(i, j) \tag{2}$$

where v is the number of rows in a block and s is the number of columns. It should be noted that the ordering of the sparks is responsible for the break up of the relationship matrix into block diagonal form. Pairings may occur only within subblocks, so the decision function may be applied to each block separately. The decision function is constructed to take into account both intrinsic parameters and extrinsic or contextual parameters. The intrinsic parameter is *width* and enters through F_1:

$$F_1(i, j) = 1 - |w_1 - w_2| \tag{3}$$

where w_1 and w_2 are normalized widths. The contextual parameters are *order* and *interference*, and these parameters enter $D(i, j)$ through F_2 and F_3, respectively:

$$F_2(i, j) = 1 - \frac{|i - j|}{P} \tag{4}$$

where $P + 1$ is the maximum dimension of the block and

$$F_3(i, j) = \left(1 - \frac{S(i) - 1}{R}\right)\left(1 - \frac{T(j) - 1}{R}\right) \tag{5}$$

where R is the maximum dimension of the block, $S(i)$ is the number of geometrically possible pairings of spark i, and $T(j)$ is the number of geometrically possible pairings of spark j. The F's are all normalized to yield values in the interval (0, 1). The weights w_1, w_2, and w_3 are experiment-dependent. Large values of $D(i, j)$ mean high probability of pairing. On the other hand, low values of $A(i, j)$ mean high probability of pairing—a low value of $A(i, j)$ means that there is low probability that the ith spark of view one could be paired with any spark other than the jth spark of view two or that the jth spark of view two could be paired with any spark other than the ith spark of view one.

By repeated application of $D(i, j)$ and $A(i, j)$, successful pairings are determined. With each determination, the relationship matrix is reduced

by one row and one column until all ambiguities are resolved. It is possible that all ambiguities are not resolved and that some are left unresolved until the linking operation.

The PAIRING ALGORITHM then passes on to the LINKING ALGORITHM a relationship matrix with almost all ambiguities removed.

D. Linking Into Tracks

The output of the PAIR program is a set of three space coordinates that form a graph. The LINK program selects the proper tree (or forest in the case of multiple events) to represent the particle tracks. The detailed tree-selecting algorithm is given in Reference 2. Briefly, edges are established between the various vertices of the graph on the basis of a number of criteria, such as Euclidean distance, number of vertices, direction of the edge, linear and helical extrapolation, and geometry of the spark chamber. Finally, the subgraph selected is the minimal connector tree that contains no circuits. The determination of the minimal connector tree is accomplished by an algorithm due to Kruskal [Reference 11].

The graph data are stored in memory in a multi-word list. Each list item contains all the necessary information about a given spark. Each list item contains seventeen words in all, including the three space coordinates of the spark, its gap number, its chamber number, the local degree of the vertex, pointers to as many as seven connecting sparks, distance and pointer to the closest spark, and distance and pointer to the second-closest spark.

Now that we have arrived at a set of coordinates that represent the paths of particles participating in the event, the data are sent on to fitting programs and programs that extract the physics information from the data.

IV. CONCLUSIONS

In order to develop more complete systems of the type depicted in Figure 1, it will be necessary to incorporate complex analysis programs of the type described in Section III above. This is by no means all of the story. The kinematic analysis programs and the hypothesis-testing programs that follow are also very complex. However, progress is being made, and physics data are being analyzed at a very rapid rate. Moreover, work is independently proceeding on the control aspects.

One development which I feel is not getting its full share of attention in these problems is in the area of displays, control consoles, and the system control languages. That is, the command posts are not being developed as thoroughly as they should be. The splendid use of graphics that we have seen in other areas could be put to excellent use in these large analysis and control systems.

REFERENCES

[1] F. SOLMITZ, Stanford Particle Physics Colloquium, April 5, 1965 (unpublished).
[2] R. CLARK and W. F. MILLER, "Computer Based Data Analysis Systems," *Methods in Computational Physics*, Vol. V, Academic Press, p. 47 (1966).
[3] W. MILLER, J. MEADOWS, A. SMITH, and J. WHALEN, "On-Line Data Acquisition with Feedback to Accelerator Control," Proc. of Conference on Automatic Acquisition and Reduction of Nuclear Data, Kernforschungszentrum, Karlsruhe, July 13–16, 1964, Session III, Gesellschaft für Kernforschung m.b.H. Karlsruhe, Germany, p. 222 (1964).
[4] A. B. SMITH and J. WHALEN (to be published).
[5] E. NORBECK, "An MPA System That Is Flexible, Reliable and Fast," Proc. of Conference on Utilization of Multiparameter Analyzers in Nuclear Physics, Grossinger, New York, November 12–15, 1962, N. Y. O. 10595, p. 56 (March 1963).
[6] W. F. MILLER, "The Role of Computers in Experimental Physics: A System for On-Line Analyzers," Proc. of Conference on Utilization of Multiparameter Analyzers in Nuclear Physics, Grossinger, New York, November 12–15, 1962, N. Y. O. 10595, p. 143 (March 1963).
[7] R. H. VONDEROHE and D. S. GEMMELL, "A Description of the PHYLIS On-Line Computing System," Proc. of Conference on Automatic Acquisition and Reduction of Nuclear Data, Kernforschungszentrum, Karlsruhe, July 13–16, 1964, Session IIc, Gesellschaft für Kernforschung m.b.H. Karlsruhe, Germany, p. 156 (1964).
[8] "The NBS On-Line System—Design for Flexibility," *Nucleonics*, Vol. 22, p. 57 (December 1964).
[9] S. J. LINDENBAUM, "The On-Line Computer Counter and Digitalized Spark Chamber Technique," *Physics Today*, Vol. 18, p. 19 (April 1965).
[10] D. HODGES, "CHLOE, An Automatic Film Scanning Equipment," ANL-AMD Technical Memorandum No. 61, November 1963 (unpublished).
[11] J. B. KRUSKAL, "On the Shortest Spanning Subtree of a Graph," *Proc. Amer. Math. Soc.*, Vol. 7, p. 48 (1956).

DISCUSSION

A. P. BATSON: Does your decision process take account of the situation where, on one stereo view, you only have one track, whereas two exist?

W. F. MILLER: We are only pairing sparks, now, individual sparks.

A. P. BATSON: I thought you used this on film.

W. F. MILLER: We did.

A. P. BATSON: Bubble chamber pictures, too?

W. F. MILLER: Spark chamber film, only! These were spark film, where the sparks have individual character, and you identify them in stereo view, first. This has considerable advantage for the next step, because many of the ambiguities which might be not obvious in one view (for example, crossing tracks) are not present in three-space; so doing the pairing operation first removes the ambiguity and permits the linking operation to go much faster.

H. J. BARNHARD: Going back to your original premise, I wonder if, eventually, this whole system doesn't go through a complete cycle. You introduce man at the bottom of the scheme, and man makes certain changes. Eventually, man goes through enough potential tries, some of

which are correct and some of which are wrong, and the machine learns which are the desirable routes to try. Then, the machine can try the available techniques and look at the problem without man's assistance.

In brief, I don't think you have carried the idea far enough. Man entered the process, and man can eventually leave it.

W. F. MILLER: In the first programmed experiment I showed, it's certainly true that he goes home, writes a program, cogitates, goes back to the lab, key-punches and runs in the paper tape, and walks away. He certainly does leave the process in many experiments, particularly those having the character of a measurement, that is, not much experimenting. But most of us still feel that the best decision box is a human, that by putting the human back in the loop you give him a chance to guide his experiment. [In a console system like Project MAC, the real hope is that, some day, a man will be sitting at the console and make a discovery; maybe it's happened already.] Then, what one expects is that a man is in control of the experiment; he sees what's happening and whether there is something that doesn't fit his model; he may come up with an invention. The experimenter has no chance of changing the experiment if the analysis is done many days or weeks later. Our experience on the Van de Graaff was that it paid to put the man back into the experiment.

A. H. GOTT: At the beginning, you made some comment about digitizing 20 to 30 events per hour with the human observer?

W. F. MILLER: Quite right.

A. H. GOTT: And 200 to 300 with your scanning equipment?

W. F. MILLER: Yes, when connected to a computer of the 3600/7094 II class. The analysis goes at the rate of less than 20 seconds per stereo view.

A. H. GOTT: Who selects the stereo views?

W. F. MILLER: We scan all pictures. Incidentally, these spark chambers are triggered by external information; so in spark chambers, if you know a little of what's happening, your success ratio is relatively high, maybe one in four or one in three. In bubble chambers this is not the case.

A. H. GOTT: You are digitizing 300 events per hour?

W. F. MILLER: The digitization of one frame takes about four seconds, that is, a stereo view, in eight seconds. So digitization without any analysis can go at a rate of about 450 stereo views per hour. The analysis rate is the limiting factor.

A. H. GOTT: The rest of the time is occupied in this?

W. F. MILLER: The analysis is overlapped, actually, on the system. Actually, they are limping along right now, doing it serially, but the provision is there for overlapping and scanning and analysis, which is something less than 20 seconds on a machine of the 3600 category. We are thinking now of a machine that will reduce this to a fraction of a second.

14
Applications of a Computing Facility in Experiments on Human Visual Perception*

DEREK H. FENDER
California Institute of Technology

INTRODUCTION

The gradual development of computer science throughout the United States and the world has been marked by a series of advances as first one discipline and then another have found it possible to weld together their own experimental techniques and methods of analysis with the potentialities of a digital data processing device. One of the more recent entrants to this field is the life sciences, and already it is possible to name many areas of biological or biomedical research where computing facilities are used as an experimental ancillary. The purpose of this paper is not to explore this aspect of the whole field of biological research but rather to concentrate on a very narrow area, the uses of a computing facility as an active participant in experiments on visual perception.

Biological experiments are usually not as well ordered as experiments in the physical sciences; small differences between one preparation and the next often mean that the investigator cannot proceed by rote but has to direct the work by a series of decisions involving future strategy; usually these are procedural or logical decisions; sometimes they may involve calculations, but always they have to be made against the harassment of a dying preparation or a tiring experimental subject. It is in a context such as this that a computer can be a powerful scientific assistant, provided only that a free flow of information is possible back and forth between the experimenter, the experiment, and the computing device.

Within the field of visual research, there is still a wide compass of methodology; the efficient design and usage of the computing system

* This research is supported by the National Institutes of Health, Grant No. USPHS NB 03627-04, United States Public Health Service.

turn very strongly on the details of these techniques, and thus it is profitable at this juncture to outline the range of experiments which might be found in a laboratory devoted to the examination of visual processes.

TYPICAL EXPERIMENTS IN VISUAL PERCEPTION

Experiments may be made at many levels; for example, a neurophysiological examination of the visual system is essentially an input-output type of experiment in which the output data consists of parameters describing the neural impulse traffic at some point in the optic pathways; the input may be stimulation by light entering through the normal optics of the eye, or it may involve electrical, chemical, or mechanical stimulation at some other point in the system. These experiments in general involve surgical procedures on animals; the duration of the experiment is of necessity short, ranging from a few minutes up to about one day, depending on the species. The possibility of replicating the experiment on the same animal over a long period of time is remote.

A second form of experimental procedure examines the motor response of the subject, or of a sub-system of the subject, to various external stimuli. As examples we might quote eye-hand coordination tasks; the stimulus is a moving target in the visual field, while the system output is the motor activity producing motion of the arm, hand, and wrist as the target is tracked with a pointer. In this case the dynamics of the limb probably represent the dominant member of the system, and thus the amount of information which can be gathered concerning the visual system is sparse. But the experiment may also be performed on a subsystem, such as the motion of the eyeball when tracking a moving target. In this case the dynamics of the eyeball do not constitute the limiting feature, and considerable information concerning visual function can be obtained from experiments of this type.

Psychophysical experiments are also powerful in the evaluation of visual function; a perceptual task is presented, and the subject's report of the situation supplies the output data for the experiment. For example, a dim flash of light near the visual threshold may be presented and the subject asked to respond when he sees it; or a fine line may be presented in the visual field and the subject asked to identify the orientation of the line as a test of resolution. This use of a subjective response always lays the experiment open to a number of criticisms [Brindley, 1960]; control presentations must always be made in order to assess the reliability of the subject. It is sometimes possible, however, to bypass the subjective response by measuring a concomitant parameter which is well correlated with the perceptual task; for example, evoked potentials recorded on the surface of the cornea or of the scalp may be used as indicators of

events entering the visual system and proceeding to the cortex. Similarly, the synkinetic movements of convergence and divergence may be used as indicators of accommodative changes to resolve a visual image.

EXPERIMENTAL CHARACTERISTICS WHICH DICTATE THE MODES OF DATA PROCESSING

While a large general-purpose computing system can in general handle any data reduction task, the process can be made much more facile if the computer, or at least the peripheral equipment, is tailored to the job in hand. In a teaching and research institution such as the California Institute of Technology, it is hardly possible to include special facilities in the central computer, for there are many users with widely diversified requirements. Instead, each user with a direct connection to the computing system provides his own special facilities in his peripheral equipment. The purpose of this section is to review the characteristics of experiments on the visual system and to outline their special requirements.

Neurophysiological experiments on the visual pathways usually use a well-defined function as the stimulus, such as a flash of light of known brightness, hue, saturation, and temporal wave form, to illuminate the photoreceptor. The output, however, is much less well defined; it may either be a small potential generated in or near the receptor itself and varying only slowly in time, or it may consist of rapidly changing repetitive potentials (nerve impulses) in the axon leading from the receptor; it is sometimes possible, by careful placement of the electrodes, to record these two potentials simultaneously (Figure 1). In either case, the response is contaminated with added noise, is not repeatable in detail on replication of the experiment, and is probably quickly adapting.

These points of themselves impose experimental problems: the noise can be reduced by averaging techniques, but so will the fine detail of the response be reduced, since this is often not accurately time-locked

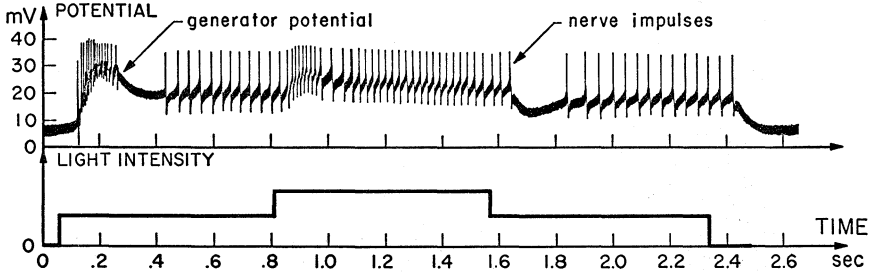


FIGURE 1. Generator potential and nerve impulses recorded in an ommatidium of the compound eye of Limulus [Redrawn with modifications from Ratliff, 1961]

to the major component. Correlation methods applied before averaging may avoid this difficulty, but the many repetitions of the stimulus which are required if an averaging process is to be applied may well cause the response to adapt right out. Adaptation, however, is a systematic process; hence allowance can be made for it, provided the complete time history of the experiment is preserved. These features indicate that the experiment should be automated: run on a pre-programmed schedule with the presentation of the stimulus and the data-gathering regimes fully integrated. If possible, the data analysis should run concurrently with the experiment, for then the adaptation of the preparation can be monitored and the repetition rate of the stimulus adjusted accordingly, or the experiment may be terminated if the preparation proves unreliable.

The output of a neurophysiological experiment is copious, even for a single channel; and in general many output channels (nerves) will be active for every receptor stimulated. Each nerve impulse is of about 2 msec duration and thus should be sampled at least every 0.1 msec if the pulse wave shape is to be accurately represented. This means a data inflow rate to the computer of 10,000 samples per second per channel, and it may well be necessary to examine 5 to 10 channels in relatively simple interaction experiments. The recording accuracy of this data is not high, and it can be adequately represented by about 10 bits; thus a number of data-points can be packed into each computer word; but even so, an experiment of this sort fills up available storage space in a computer at a great rate.

We do not know for certain where the information resides in the train of nerve impulses which constitutes the output of a neurophysiological experiment, but it is presumed that the important parameter is the time sequence of the impulses. Many forms of analysis are based entirely on the time history of the nerve impulses; hence, it is often adequate to extract this parameter alone from the experimental output in peripheral equipment before transmission to the central processor.

Already it is evident that a hierarchy of control and analytical devices is necessary for experiments on the visual pathways. First a local control stage is required at the experiment itself; this equipment is particular to each experiment and is responsible for the detailed sequencing of the experimental procedure, under command from some higher level in the data processing chain. Next we require a data abstraction level, possibly common to all biological experiments. This permits a certain pre-processing of the data, so that only the information which is required for the final analysis is sent to the central processor. The data abstraction level must also have an information route to the experimental control equipment so that it can interact in the presentation of the stimulus and primary analysis of the data. The last stage is represented by the general-purpose

computing facility; this must be available on-line at least for data-gathering purposes, for a biological preparation deteriorates from the moment it is made and cannot "wait its turn" on the computer. If on-line analysis is also possible, then the power of the experimental system gains considerably, for now direct feedback of information is available from the computer as an immediate experimental control and as a guide to strategy.

Stimulus-*versus*-motor-response types of experiment can be handled with equipment of this sort but demand a few refinements. The input is well defined, but for a human subject the interaction between sense modalities may be subtle. For example, the reaction time for fixating on a light suddenly switched on in an empty visual field is shortened if a noise occurs at the same time. Great care must be taken that the pre-programmed stimulus unit excites only the sub-system under examination. Similarly, although the output analysis may easily be confined to a single channel, the response may well occur in several; in the fixation example quoted above, the eyes turn to look at the target—the tension in the extraocular muscles might thus be the output. But in general the head also turns; flexure of the neck should therefore be eliminated by fixing the head or should be examined as well. It is thus necessary to record all sub-system responses elicited by a certain stimulus and to look for their interactions. Biological systems may even have several modes of response for the same external stimulus. This is shown in Figure 2, which records the eye movements* of a subject when following a target moving sinusoidally from side to side. In the lefthand section of the diagram

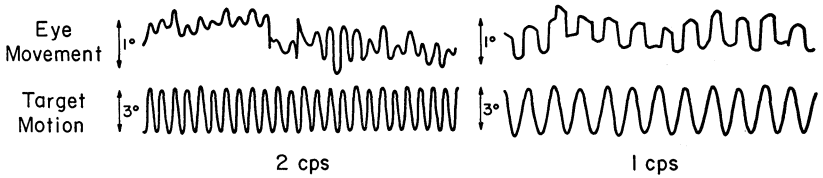


FIGURE 2. Different modes of eye movements used by subjects in horizontal tracking of moving stimulus

the subject's direction of gaze follows the target smoothly (with some noise), but in the righthand portion the response consists largely of saccades. The analytical system must be aware that such switching of modes is possible and should be ready to apply the analysis appropriate to each mode.

* Most of the examples quoted in this paper refer to human eye motions. The method of measuring these movements is treated shortly in the appendix, which also explains some of the terminology.

Psychophysical experiments pose a further problem. In this case the subject is presented with a visual task and asked to respond to a question such as "Did you see a flash of light?"; his responses constitute the output data of the experiment. The difficulty here is to ensure that the subject and the experimenter both have the same understanding of the task, i.e., to standardize the mental sets of experimenter and subject. Cornsweet and Pinsker [1965] have argued that this is achieved if the subject gets immediate feedback concerning the correctness of his responses; from this he will be able to deduce the interpretation placed upon the task by the experimenter; thus after an appropriate training session they will experiment from a common viewpoint. In an automated experiment it is thus essential that this feedback of information should be handled by the computing complex. Psychophysical tasks always represent a decision process in the subject and should be analyzed as such; this means that an interspersal of false presentations of the stimulus must be made in order to establish the reliability or criterion level of the subject.

COMPUTING FACILITIES REQUIRED FOR BIOLOGICAL EXPERIMENTS

Bearing in mind the points outlined above, we can now draw together some of the requirements for a data processing device for experiments in the neurophysiology and psychophysics of vision. The short life of some biological preparations or the restricted tolerance times of human experimental subjects makes it essential that the experimenter has immediately available an on-line means of data collection. Analog magnetic tape is not highly satisfactory in this respect, for it halts the flow of data analysis. It is better if the information can be stored directly in the computer memory, for then analysis may proceed concurrently with data collection or at least immediately after it. This is a real necessity in biological work, for the experimenter needs an immediate display of his raw data (or derived parameters) as a guide for experimental strategy—biological preparations are not sufficiently reproducible from one animal to the next for this feedback of information to be unnecessary.

Once the experimenter has established his preparation and formed an experimental plan, then he must be provided with a hierarchy of data abstraction stages carrying out progressive compression of information before it reaches the central processor. Biological experiments are so prolific in the production of data that it is only in this way that one can avoid saturating any computing system with input data; this would slow down the speed of computation and reduce the value of any resulting feedback to the experimenter. But it is also desirable that the raw data should be stored in its original form, for hindsight is a most powerful experimental tool, and the experimenter should be able to recover and

work over data gathered in previous experiments. Finally, feedback to the experiment should be available from all levels in this hierarchy: from the lowest levels to control the experimental regime, from higher levels to give guidance to the experimenter or to the experimental subject, and from the highest levels to monitor the outcome of the experiment.

COMPUTING SYSTEM FOR BIOLOGICAL RESEARCH AT THE CALIFORNIA INSTITUTE OF TECHNOLOGY

There are obviously many ways of implementing the computing strategy outlined above. The method adopted at the California Institute of Technology has been described by McCann and Fender [1964]. The present computing complex consists of an IBM 7040 and an IBM 7094 interconnected through switchable tape units, disk files, and a trap-line which gives direct core-to-core transfer of small amounts of information between the computers (Figure 3); broadly speaking, the 7040 handles all the administrative side of a program while the actual computation is carried

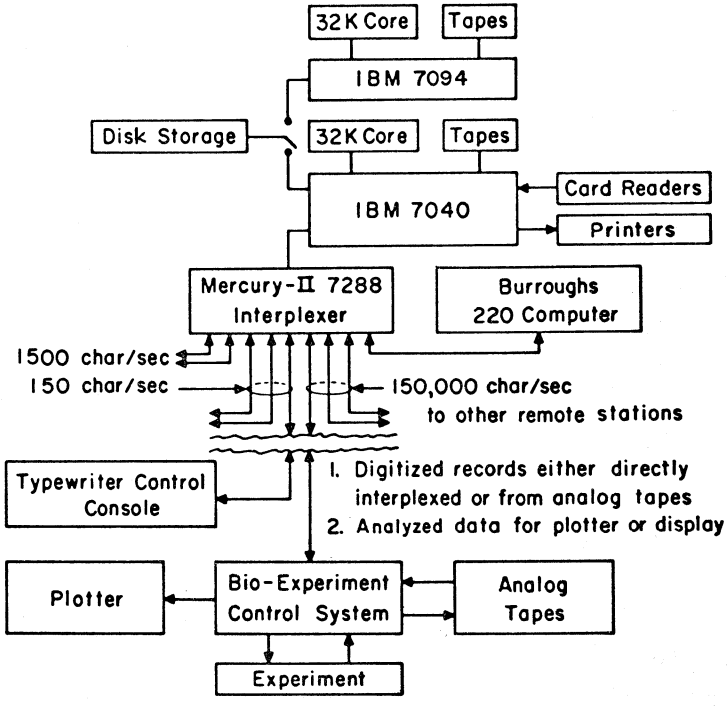


FIGURE 3. Schematic diagram of computer layout for research in visual perception

out on the 7094. Access to the computing system is through a multiplexor unit on a time-sharing basis, and the 7040 is always available for on-line data collection. Control of the computer for this purpose is effected from remote typewriter consoles located at the sites of the experiments. A typical experimental routine would be: (i) enter the equivalent of a few control cards on the typewriter console; this transfers the appropriate data collection programs from the disk files to part of the 7040 memory; (ii) transmit data; this will generally be stored on digital tape after a certain amount of editing by the central processor; (iii) at the conclusion of data gathering, enter the equivalent of an "execute" card on the typewriter console, naming the analysis program to be used; this initiates a train of events as follows.

The job is allotted a priority depending upon its predicted running time and amount of output; normally, biological experiments fall within the highest priority. The job is then entered at the bottom of the first-priority queue. When the job reaches the head of the queue, the 7040 transmits a disk-map of the data analysis program (which already exists on the disk files) through the trap-line to the 7094; this in turn reads the program from the disk, the data from tape, carries out the analysis, and then returns the results to the disk. The 7040 receives a disk-map of the results from the 7094 through the trap-line and handles the output via high-speed printers, the typewriter console at the experiment, x-y plotters, or other suitable media. The average turn-around time for this process at present is 8 minutes.

Information is transmitted from the experiment to the computing complex by a special-purpose computer (the Biological Systems Data Terminal). This device can operate in a number of primitive modes; the simplest merely records analog data from the experiment together with a local system clock track on a multichannel FM tape recorder. Alternatively, the analog data from the experiment may be passed through an A/D converter, gated by the local system clock, and then transmitted to the 7040. Both of these operations may be performed simultaneously. Signals derived from the analog tape may also be played back through the Data Terminal, converted to digital form, and then transmitted to the 7040.

In these primitive modes the whole of the experimental data is transmitted to the central computer; but it has been pointed out earlier that a biological system can easily saturate even a large computer if all of the data is transmitted; thus certain levels of data abstraction are needed before the information reaches the central processor in order to avoid this possibility. To provide this facility, the Data Terminal contains a large number of logical element modules. The experimenter has access to these through interchangeable patchboards; he can then wire up the Data Terminal to perform any data abstraction his ingenuity can devise

with the logic available. Examples of this will be given in the paragraphs which follow. These logical elements may also be used to derive timed control signals from the system clock for programming the experiment or to transmit information concerning the data flow back to the experimenter.

TYPICAL EXPERIMENTS IN VISUAL PERCEPTION

The realization of some of the desiderata considered above can best be illustrated by the case histories of a few experiments in visual perception.

Fixation Studies

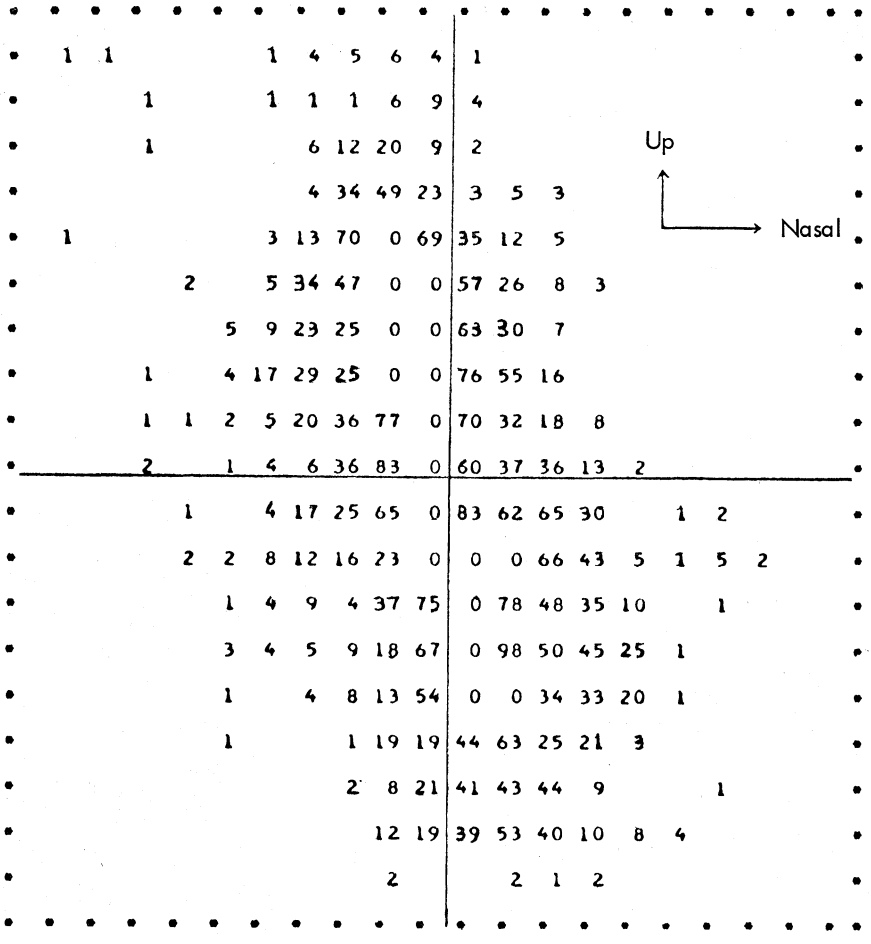
The human eye is never at rest; even in examining a small point object closely, the direction of gaze sweeps around the fixation area with involuntary motion and very rarely points directly at the target.

The underlying reasons for these apparently spontaneous movements are of considerable interest for theories of muscular activity and of the neurophysiology of vision; insight into both of these processes can be obtained from a consideration of the statistics of the motion.

Both the horizontal and vertical motions of the visual axis of each eye are needed for this study; thus four channels have to be recorded. Except for the flicks, which last about 30 msec, the eye movements are of low frequency, so sampling the analog data at 20-msec intervals is adequate. The logical modules in the Biological Systems Data Terminal are therefore set up to multiplex these four channels through the A/D converter and transmit them to the 7040 at a total rate of 200 data-points per second—a very low figure compared with the capabilities of the system. The experiment is automatically timed for a two-minute run by the Data Terminal, which also inserts a displacement calibration step in each eye-movement channel at the beginning and end of each experimental run.

The primary analysis carried out on this data consists of unpacking the multiplexed data into four separate channels, scaling each one from the calibration pulses, and then generating a two-dimensional histogram similar to Figure 4 for each eye [Fender, 1964*a*]. The numbers printed on this diagram show the amount of time the subject spent in the examination of each cell of a rectangular network centered on the target. Supplementary analyses examine this distribution to see if it can be partitioned into sub-distributions. In general this proves to be true; a search is then made for the transfer mechanisms between these areas; flicks are usually found to be responsible.

From the description of the experiment it seems that there is no man-machine interaction; the computing complex is running the experiment entirely on its own. This is true; the experimenter is performing another



Cells in fixation histogram are 2.0 min arc square.
The squares of the deviations are: horizontal–13; vertical–65; and cross– –14.
The total number of points is 5,787.
The temporal skew angle is 14° from the vertical.
The ratio of the axes of the ellipse is 2.7:1.

FIGURE 4. Two-dimensional fixation histogram—left eye
[Record obtained by Dr. G. W. Beeler]

experiment concerned with perception on the same subject at the same time. This is cooperation of a high order, for in psychophysical experiments the rate of acquisition of meaningful data is pitifully slow, and any artifice which extracts more information during the period of subject viability represents a real gain.

Dynamics of Flicks

The dynamics of the flick motion is of interest, for it permits *in vivo* measurements to be made related to the biophysics of the eye muscles, a study which can normally only be done *in vitro*.

For this purpose, horizontal and vertical components of eye movements from one eye are recorded on two tracks of the analog tape recorder. At the same time, electronic equipment at the site of the experiment detects the occurrence of flicks in each channel and records square pulses on two corresponding channels; the polarity of the pulses indicates the direction of the flicks—up or down, temporalward or nasalward. The Data Terminal is now interconnected so that when the analog tape is played back into the computer, multiple read-heads are used; those which read the pulse tracks are merely used to sense the rising edge of a pulse and then to transfer control to another head placed earlier in time on the track containing the wave form of the flick. This head reads the analog signal and transmits it through the A/D sampler (at 1,000 samples per second for a fixed number of samples) to the central computer; thereupon the system returns to the head which is monitoring the pulse track, and so on. The lead time and the number of samples are chosen so that the whole of the profile of the flick is transmitted.

On entering the computer, the flicks are sorted—by subject, by left or right eye, and by visual task—from information inserted at the console before each experiment; by direction (temporalward or nasalward, up or down) from the flick detector circuits; and by type (magnitude, damped or overshot, spontaneous or target-following) from programming. Once the flick has been categorized, it is matched by a chi^2 test against the average profile of all other flicks collected so far in that category. If it meets a criterion, it in turn is averaged into the template; if it does not, then it is printed out for the experimenter's examination; a reject is shown in Figure 5.

When any one category has collected 100 flicks, data collection in that class is terminated by the program, and the following analysis is initiated. The phase portraits (displacement against velocity) of all the flicks in a class are overprinted as a two-dimensional histogram (as in Figure 6) where the figures now indicate the number of trajectories which go through each cell in the phase plane. A smoothed crest-following routine converts this to a single trajectory from which the natural frequency and damping of the eyeball as a function of time can be evaluated; this permits the changes in tension and viscosity of the eye muscles during the flick to be calculated [Fender, 1964*b*].

The form of interaction between man and machine illustrated by this example is the interchange which takes place whenever anything abnormal

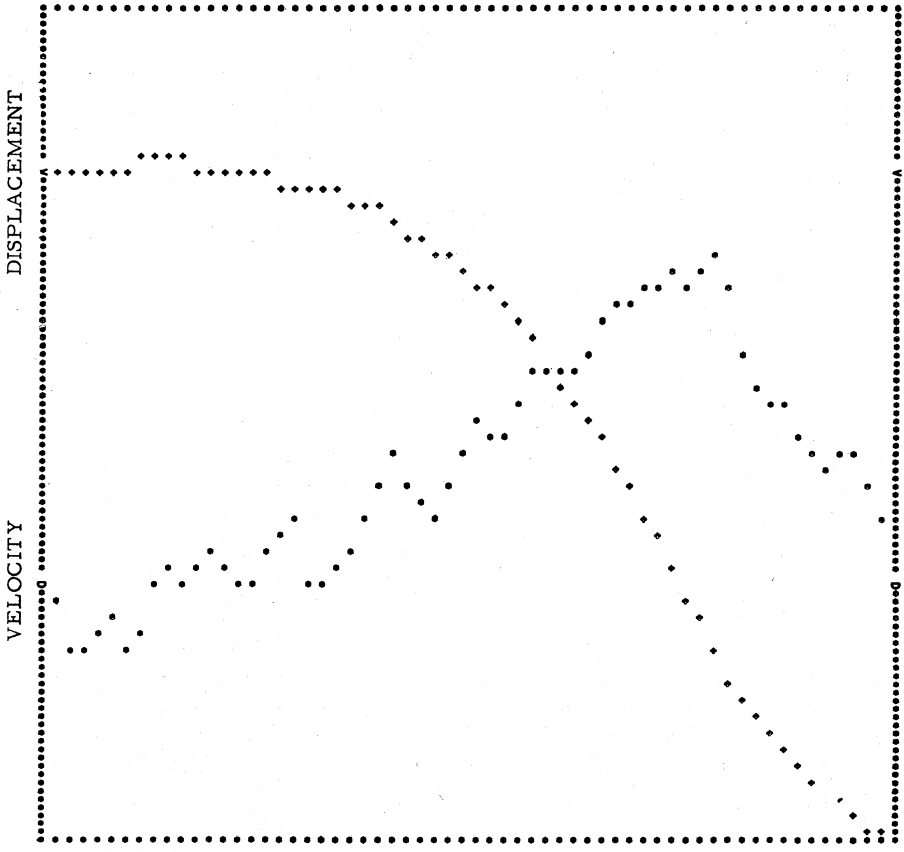


FIGURE 5. The upper part of this diagram shows the side-to-side displacement of the visual axis plotted against time during a saccade (up = left). The velocity is illustrated in the lower portion of the diagram. The reason for rejection is the high initial value of the velocity. Total time illustrated is 40 msec.

happens; one of the dangers of computerized biology is that information which would normally be examined visually disappears into the maw of the computer and only emerges in a highly digested form—anything abnormal (which may therefore be of considerable interest) stands a high chance of being lost in this process. It is important that the data analysis programs should be so written that anything abnormal is detected and referred back to the experimenter for his further consideration.

[*Text resumes on next lefthand page*]

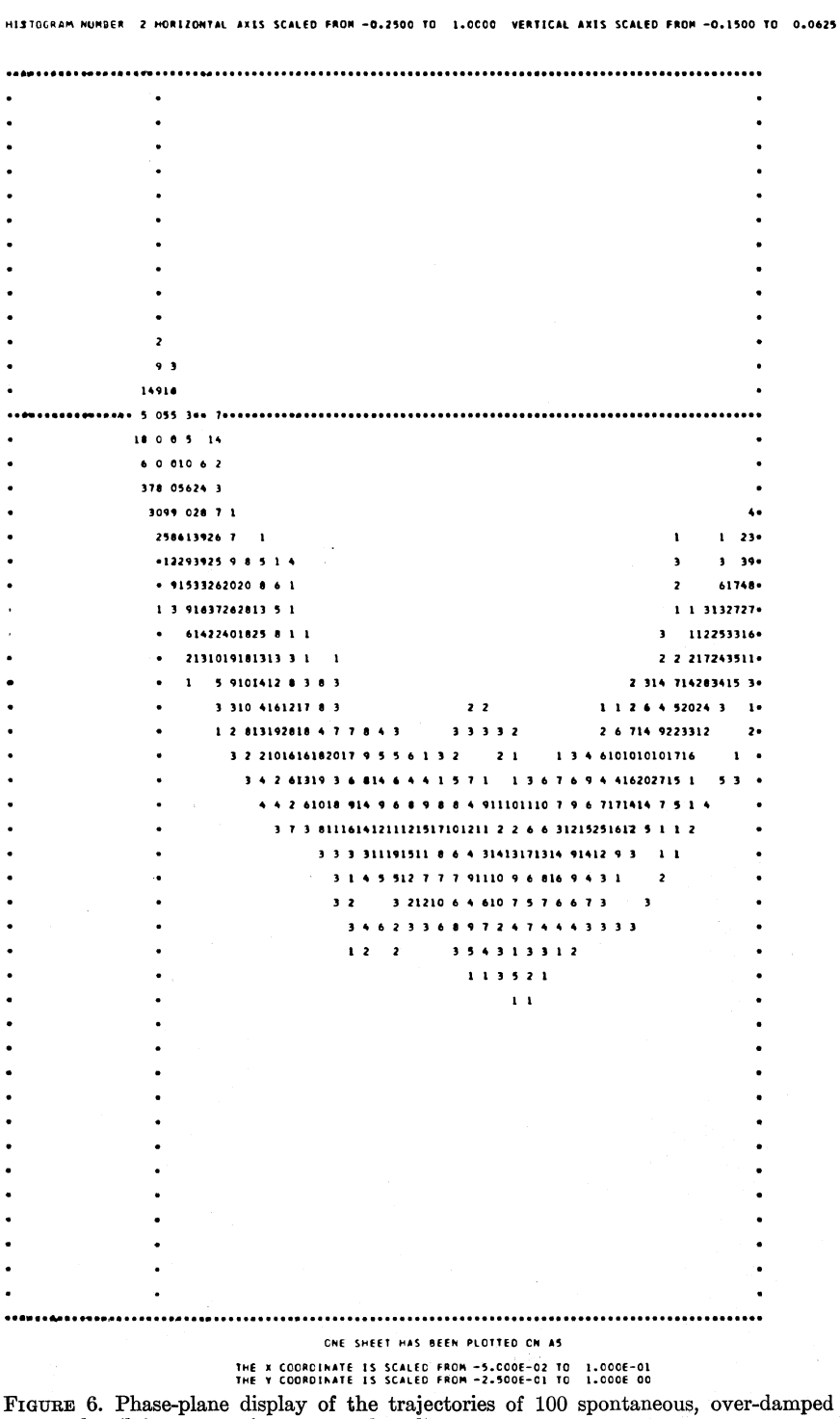


FIGURE 6. Phase-plane display of the trajectories of 100 spontaneous, over-damped saccades (left eye moving temporalward)

Suppression of Vision During a Saccade

The last experiment to be described by way of illustration turns on the question of whether vision is suppressed during a saccade, for we certainly do not get the impression of great movement of the visual scene during a flick, even though the image is moving across the retina with considerable velocity. There are a number of theories to account for this apparent stability of the visual world; one of them is that the threshold for brightness discrimination is elevated considerably during a flick, sufficient to suppress perception for a short period. One way to test this is to flash a light (which is suprathreshold in normal vision) during the 30-msec period of a flick and to test if the light is still visible. It turns out that the effect which we are seeking begins some 60 msec before the flick and lasts about 100 msec after it. Events occurring during and after a flick can of course be explored, using normal electronic techniques to trigger the flash from the flick, but the period before a flick presents a rather different problem. Fortunately, the flicks occur at a mean repetition rate of about 2 per second with a normal distribution about this period—so that if the flick detector circuits are used to sense one flick and then to trigger the flash about 500 msec later, the flash will coincide approximately with the next flick. A histogram of times of occurrence of the flash with respect to the next flick would now be normally distributed, whereas we require the same number of observations (50) in all cells 10 msec wide from 100 msec before the flick until 100 msec after it.

It is at this point that the computer begins to interact with the experiment. The time constant of the flash-generating circuit is set to be 100 msec shorter than the mean interflick period for the subject. The Biological Systems Data Terminal programs the experiment in two-minute runs as follows. At $t = 0$, it arms the electronic equipment at the experimental site; this equipment then generates a flash following the first spontaneous flick. At $t = 2$ sec, the Data Terminal illuminates a lamp in the peripheral visual field, inviting the subject to RESPOND; the subject presses a key reporting "seen" or "not seen." At $t = 6$ sec, the Data Terminal, in conjunction with the local electronics, lights another sign announcing CORRECT or WRONG; some false negative presentations are used in order to test the reliability of the subject and to keep his motivation at a high level, but this is only effective if he receives feedback concerning the correctness of his judgments. From $t = 7$ sec to $t = 10$ sec, the subject prepares for the next presentation, and then the sequence repeats. We normally obtain 96 observations in the 8 runs which constitute an experimental session.

The logical units of the Data Terminal are wired so that only the times of occurrence of the pulses of light, the time of the associated flick, and

the subject's report are transmitted to the computer. At the end of each experimental session a histogram is formed of the times of occurrence of the flash with respect to the flick; a normal distribution is fitted to this histogram. The program then calculates the change which is required in the time constant of the flash generator in order to move the mean of the normal distribution to 100 msec before the flick. This information is signalled back to the experimenter, and an appropriate adjustment is made. This cycle is repeated until the histogram cell at 100 msec before the flick contains at least 50 entries; now the computer indicates the correction necessary to fill later cells in the histogram, until at least 50 observations have been obtained in each cell up to the time of occurrence of the flick. At this point the computer indicates that the experimental regime should be changed, and the flash is merely triggered at pseudo-random integral multiples of 10 msec after the present flick until all the cells in the histogram up to 100 msec after the flick are filled, whereupon the experiment is terminated. It remains then to calculate and graph the probability of seeing the flash stimulus during each interval, as shown in Figure 7. It will be observed that the chance of seeing a flash of light

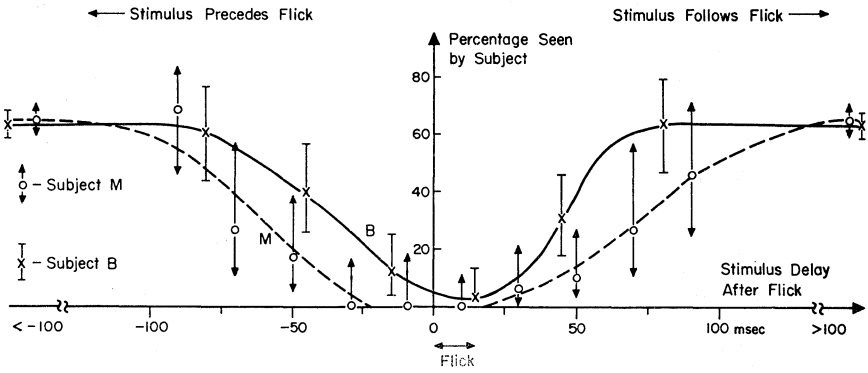


FIGURE 7. Plots showing time course of brightness threshold change relative to time of flick initiation. Vertical points around data-points indicate 99 percent confidence intervals. [Record obtained by Dr. G. W. Beeler]

is reduced if it occurs at about the same time as a flick, but if the curves of Figure 7 are cast into threshold measurements, it will be found that the threshold rises by only about 0.5 log units during a flick; this is insufficient to account for the apparent stability of the visual world.

CONCLUSION

This paper did not set out to cover man-machine communication in the form of verbal messages, written text, or diagrammatic information, but rather the form of communication which is required between members of a research team. In the work which has been described here, the computer is one member of this team. It is possible to regard the computer as playing a rather lowly role in the research group—it is the member which records the data, makes quick calculations to see how the experiment is going, jogs the experimenter's memory when some experimental condition has to be changed, and presents for his evaluation some atypical event which may have occurred amongst the data.

In biological or psychophysical research, however, this is not a trivial role. Rarely, in work of this nature, can the theorist set up such a clear-cut dichotomy that it can be resolved by one crucial experiment; painstaking observation and the sifting of limitless information are the normal occurrences. With a few notable exceptions, the general participants in biological research are not skilled in the techniques of handling numerical data; possibly this is one of the reasons why quantitative biology is such a young member of one of the oldest sciences.

APPENDIX

Human Eye Movements

The majority of the remarks in this paper have been concerned with the use of a computing facility as an experimental aid in the examination of the interaction between scanning patterns of eye movements and various forms of visual perception. A few words are therefore appropriate on the techniques of eye-movement recording and transmission of data to the computer.

The subject wears a tightly fitting contact lens (Figure 8) which follows the movements of his eyeball quite faithfully. A stalk is attached to the lens and protrudes between the eyelids (Figure 9). A small mirror is attached to the end of the stalk, and a beam of light may be reflected from this mirror to give direct photographic recording of the eye rotation. This is useful for display purposes; but, for analysis, the information concerning eye position is obtained in another way [Byford, 1961]. Two small medical lamps are mounted around the periphery of the mirror (the power leads can be seen in Figures 8 and 9). One lamp shadow casts a straight edge onto a photomultiplier tube mounted as shown in Figure 10. Horizontal motion of the lamp changes the position of the shadow and so generates a voltage which is proportional to side-to-side rotation of

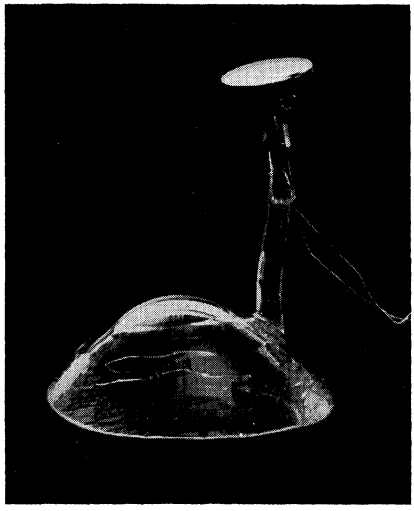


FIGURE 8. Contact lens used for eye-movement recording

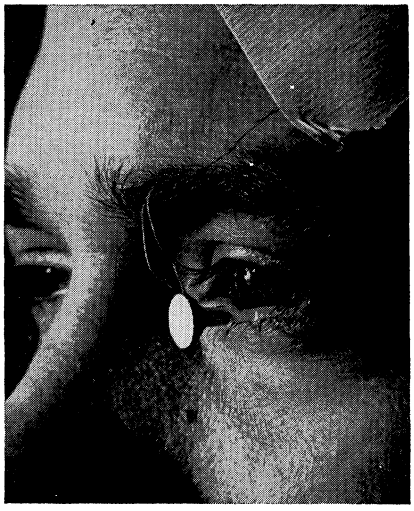


FIGURE 9. A subject wearing the special contact lens

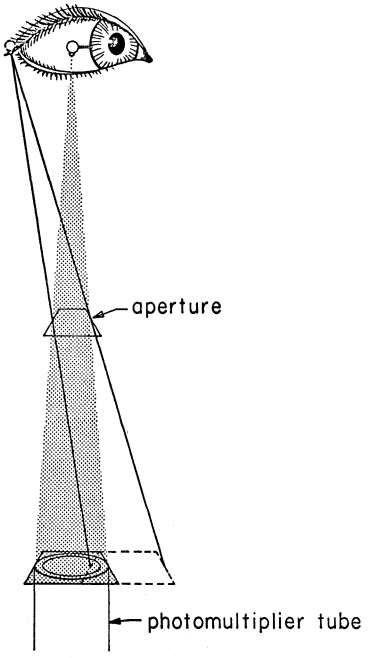


FIGURE 10. Contact lens and photomultiplier used for recording horizontal component of eye movements

the visual axis. A similar tube mounted horizontally and energized by the other lamp on the mirror records up-and-down motion of the visual axis. The voltages arising from the photomultipliers are digitized and transmitted directly to the computer by the equipment described previously.

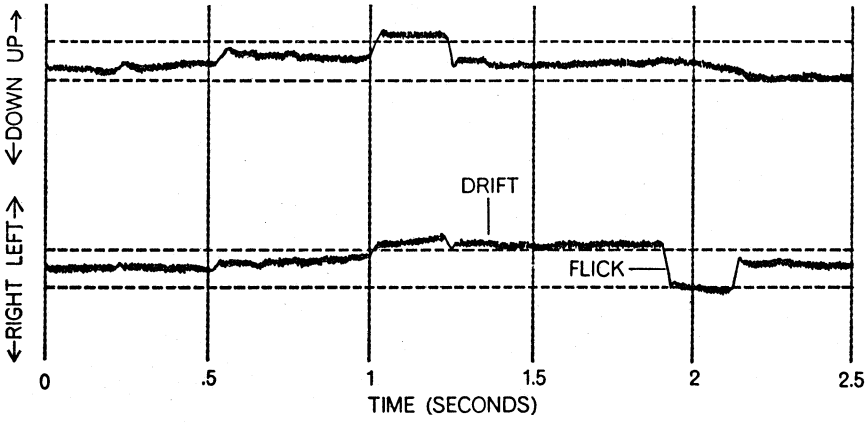


FIGURE 11. Eye movements during fixation. The pairs of horizontal dotted lines represent displacements of 6 minutes arc

Typical eye movements when the subject is fixating a small-point target are shown in Figure 11. The characteristics to be noted are slow changes in the position of the visual axis, called "drifts," and sudden impulsive changes lasting about 30 msec, called "saccades" or "flicks."

REFERENCES

BRINDLEY, G. S. 1960. *Physiology of the Retina and the Visual Pathway.* London: E. Arnold, p. 298.

BYFORD, G. H. 1961. The movements of human eyes: electronic recording techniques. *Brit. Comm. Elec.*, **8**, 334.

CORNSWEET, T. N., and H. M. PINSKER. 1965. Luminance discrimination of brief flashes under various conditions of adaptation. *J. Physiol.*, **176**, 294.

FENDER, D. H. 1964*a*. Control mechanisms of the eye. *Scientific American*, **211**, 24.

FENDER, D. H. 1964*b*. Techniques of systems-analysis applied to feedback pathways in the control of eye movements. In *Homeostasis and Feedback Mechanism* (Society for Experimental Biology symposium number XVIII), ed. G. M. HUGHES. Cambridge, Eng.: Cambridge University Press.

MCCANN, G. D., and D. H. FENDER. 1964. Computer data processing and systems analysis applied to research on visual perception. In *Neural Theory and Modeling*, ed. R. F. REISS. Stanford: Stanford University Press.

RATLIFF, F. 1961. In *Sensory Communication*, ed. W. A. ROSENBLITH. New York: Wiley.

DISCUSSION

QUESTION: How long is it practical for a human subject to wear the contact lens?

D. H. FENDER: Half an hour, at the most. We use very tightly fitting contact lenses; this is against the best optometric practice. Since the lenses fit so tightly, they interfere with the limbal circulation of the blood and with the oxygen exchange between the cornea and the atmosphere; all of this produces an increased water uptake in the cornea, so as to thicken it. The cornea is built up of parallel layers of collagen fibers, spaced much closer together than the wavelength of visible light so that a transparent film is formed. Too much water in the system moves the collagen fibers apart irregularly, and the cornea becomes turbid.

15

Man-Machine Interaction in Man-Machine System Experimentation*

G. H. Dobbs
System Development Corporation

[ABSTRACT]

In recent years System Development Corporation has been conducting a series of man-machine system experiments in two large-scale laboratory facilities designed for this purpose. The majority of the man-machine system experiments conducted have been characterized by the fact that groups of people are required to interact with or are supported by automatic information-processing systems usually as a team or organization in order to accomplish a mission or achieve an objective. Typically, the information-processing tasks performed in reaching experiment objectives require user access to large data bases. Historically, fixed procedure approaches to the processing and utilization of data-based data has limited the user's ability to react to, control, and modify his information-processing environment. As a result of recognizing this limitation, SDC has been developing and experimenting with a variety of software techniques and aids designed to give individuals the capability to design, control, and change the user system interface in near real-time. To date, these technologies have concentrated on two major facets of the user system interaction problem:

1. *The Data and Information Input Problem*
   It has become clear that techniques that allow an unsophisticated user to describe, define, and organize his data without resort to a computer specialist are vital. Considerable effort has been devoted to the development of languages and automatic processing aids to facilitate this kind of user/system communication.
2. *Information Access and Presentation*
   Experience in the laboratory has indicated that use of information is highly personal. In addition to the requirement for information need to be matched to the task or function, there is a need to match the technique of information access and presentation to individual users. To this end SDC has under development

* The manuscript of this paper was not available at the time this book was published.

several programming aids allowing the user a high degree of control over the accessing, retrieval, and presentation of data from structured data bases.

Experience to date in the laboratory indicates that although improvements are being and will be made in terms of easier physical access to and interaction with information processing systems, dramatic improvement in intellectual access and interaction is likely to come from the application of next generation software technology. This technology will assume, as opposed to the algorithm tradition of computation and data processing, that few fixed procedures exist for solving a potential user's problem. These technologies will attempt to provide tools, therefore, which will be useful in supporting heuristic approaches to problem solving.