Teaching Physiology Through Interactive Simulation of Hemodynamics

by

Timothy L. Davis

B. S. Computer Science, Indiana University (1985)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1991

Author_____
Department of Electrical Engineering and Computer Science
February 11, 1991

Certified by_____
Roger G. Mark
Professor of Electrical Engineering
Thesis Supervisor

Accepted by_____
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

Teaching Physiology Through
Interactive Simulation of Hemodynamics

by

Timothy L. Davis

Submitted to the Department of Electrical Engineering and Computer Science
on February 11, 1991, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

This thesis describes the development of a dynamic, interactive computer simulation for teaching hemodynamics. This work is an extension of the original effort begun at M. I. T. in 1984 to apply computer simulation of hemodynamics to teaching physiology. The basis of the model is a linear, time varying six compartment approximation of the cardiovascular system. The left and right ventricles, systemic arteries and veins, and pulmonary arteries and veins are modelled as capacitative compartments linked in a closed loop by resistive conduits. Blood is pumped through this circuit by time-varying left and right ventricular elastances connected through one-way flow valves.

The original cardiovascular simulator generated enthusiasm and was well-regarded by students and faculty alike, but the mathematical model was not verified, the user interface did not utilize modern workstation technology, and the simulation lacked a model of the important baroreflex control system. My objectives were fourfold:

1. Update the model and verify that it is a valid description of physiology, in that its parameters have solid experimental bases and the resulting simulation variables take on appropriate values as compared with published norms.
2. Design, implement, and validate a baroreflex model using a moving-average proportional controller.
3. Design and implement a user-friendly graphical interface to the model utilizing the X Window System.
4. Evaluate the teaching effectiveness of the new version of the simulator in the context of an undergraduate quantitative physiology course and a medical school course.

These objectives were addressed separately.

Model verification Most of the parameters were found to be adequately modelled as linear over the range of interest, and the pressures, volumes, and flows for each compartment were within the range of normal for an adult human; however, there were a few parameters which deserve to be modelled more carefully. Each compartment needs minimum and maximum volume limits in order to be physiologically accurate; this is especially important for the right ventricle, which fills to full capacity in the normal regime. The systemic arterial capacitance is sufficiently nonlinear over the physiologic range as to change the expected pulse pressure. The pulmonary resistance should be modelled as a vascular waterfall instead of a linear resistance. These suggestions have not yet been acted upon.

The time-varying ventricular elastance function was drawn from elastance measurements in dogs. It was recently found that the isovolumic contractility index for pressure $((1/P)\,dP/dt)_{max}$ is easily related to the elastance as $((1/E)\,dE/dt)_{max}$; this provides a handy reference for the slope of $E(t)$ during

isovolumic contraction. Unfortunately, value of this index for the chosen $E(t)$ is only 63% of the reported average. In addition, the time-varying elastance model of the ventricle does not account for velocity of muscle fiber shortening or the tensions developed, and thus can lead to supra-physiologic speed or force of contraction under certain conditions.

Implementation The mathematical model was implemented with a variable-timestep Runge-Kutta simulation, in order to minimize execution time while maximizing accuracy during periods in which short time constant processes were active. A set of parameter update constraints allows for dynamic parameter modification during the simulation while conserving blood volume. Examination of average pressure, volume, and flows and with pulsatile waveforms reveals good correspondence with standard values for humans from the physiology literature.

Baroreflex model A simple model of the baroreflex was developed which is composed of a bank of four finite impulse response linear filters to provide feedback gain from arterial pressure to control heart rate, maximum elastance, arteriolar resistance, and venous filling volume. The open-loop gains and impulse responses were established independently from the literature, and the closed system tested for closed-loop gain, transient response, and stability. Although the simulation appears to have more underdamped ringing than is physiologically accurate, most of the remaining parameters were within one standard deviation of reported average responses, except in the case of sympathetic withdrawal: the simple linear model of the baroreflex does not account for the increased delay due to neurotransmitter lingering at the nerve terminus for sympathetic withdrawal; this difference is noticeable in the closed-loop behavior.

User interface A user interface was developed for the X Window System, Version 11 Release 4, which utilizes the X Toolkit and Athena Widgets for generic interactions. Specialized X Widgets were developed for dynamic graphical parameter modifications, and dynamic display of simulation data. A circuit diagram interaction window links the structure of the model directly with the parameter and variable choice procedures for clarity of operation.

Student utilization Analysis of the utilization of the program was made for two populations of students, a group of undergraduate engineering students and a group of M. D. and Ph. D. candidates in a separate course. Usage logs provided information on the amount of program usage, and questionnaires assessed student opinion. The responses from the questionnaires were statistically positive from the first group of students, and overwhelmingly positive from the second group of students. The students from both groups agreed that the cardiovascular simulator exercises were useful, and that they helped in understanding the relationships between hemodynamic variables.

Thesis Supervisor: Roger G. Mark
Title: Professor of Electrical Engineering

Acknowledgements

I would like to thank Dr. Roger Mark for his valuable and insightful advice through the course of this project, and for his generous contributions of time and resources. I also wish to thank Bob Sah and George Moody for their original work on the cardiovascular simulator, and Kevin Clark, Chisang Poon, Phil Saul, Ron Berger, and Marvin Appel for their insights into models of the baroreflex. I salute the students of *6.022J* and *HST–090* for their relentless search for program bugs, their helpful suggestions, and their patience.

Lastly, I want to thank my wife and parents for their boundless understanding and support, and for being all that they know they are to me.

This research was supported in part by grants from (in temporal order) Project Athena, the Isadore and Bertha Gudelsky Family Foundation, the School of Engineering courseware development fund, and a Laboratory Graduate Fellowship from the U. S. Air Force School of Aerospace Medicine, San Antonio, Texas.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis describes my efforts to produce a user-friendly computer implementation of a mathematical model for teaching hemodynamics. Integrative research on hemodynamics has produced many models of the heart and vascular system, which for the past thirty years have helped to codify knowledge about hemodynamics and have fueled continued research in the field. One such model has been used for the past six years to teach the principles of hemodynamics in cardiovascular physiology and pathophysiology courses at M. I. T. in the Harvard-M. I. T. Division of Health Sciences and Technology.

The original effort here to create a computer teaching simulation of hemodynamics developed by Sah and updated by Moody [54] generated much enthusiasm about computer modelling for teaching hemodynamics and produced respectable numerical results, but had a primitive, menu-driven user interface by modern standards, resulting in a relatively poor level of interaction with the model. In addition, the model parameters had not been fully verified, the results of simulation were not validated, and the simulation lacked a model of the important baroreceptor reflex control system. My objective was to produce a high-quality user interface, to validate all model parameters, and to analyze the resulting simulation data, comparing each to the physiology literature to ensure that data from the simulation mimicked data from a normal human as closely as possible. I also added a model of the baroreflex control system as an additional area of exploration for students. The result is a piece of teaching software which uses an X Window System user interface and a pseudo-real-time simulation to provide intuitive interaction to a dynamic system with a "virtual patient" appearance.

1.1 Goals of physiological modelling

Simplified models are useful in the analysis of complicated biological systems. Comparison of these models to the systems of interest provides information on the extent to which their behavior is quantitatively understood. The cardiovascular system is particularly suited to modelling attempts because of its dependence on well-understood hydraulic principles and because of the wealth of understanding about the various constituent organs involved in blood transport.

A model-based understanding of the fundamentals of blood flow dynamics, or *hemodynamics,* is important for diagnosis and management of diseases of the cardiovascular system, including coronary artery and heart muscle dysfunction, valvular disorders, pulmonary disease, and systemic vascular disease. There is hope that research into models of the cardiovascular system will clarify understanding about these conditions, leading ultimately to improvements in treatment.

The cardiovascular system is a distributed system, composed of blood with varying nonlinear viscous and inertial properties interacting with vessels of complex morphology, nonlinear elastic walls, and multiple levels of homeostatic systems controlling many distributed properties. A complete description of this system would be incredibly complex and could not be accomplished in closed form. However, the use of linear approximations and lumped parameters allows a much simplified description in terms of linear ordinary differential equations. Many such models have been used to explain various aspects of the cardiovascular system [4]. Such models provide a concrete, analytical framework as a basis for understanding the full system.

1.2 Application of network theory to cardiovascular modelling

Restricting oneself to lumped parameter models allows the application of powerful tools of elementary network theory to the analysis of the biological system and to the synthesis of new models; in this case the process of modelling becomes a matter of measuring component properties and representing those properties as parameters for constituents arranged in a network. If restricted further to linear components, the network will obey the well-understood laws of linear systems theory, allowing a larger and more powerful set of tools to be used for its analysis. Control theory is also important in the understanding of biological control systems [43].

1.3 Physiological modelling as a pedagogical device

Models of biological systems serve several important purposes:

- They direct further analysis efforts. Whenever there is a departure of observation from the prediction of a model, the model must be improved. Such improvements come at the cost of additional inquiry into the nature of the biological system.

- They provide a map of current understanding. In areas where a model agrees with observation, the biological system is relatively well understood; for example, some elements of the biological system may be substantially linear and relatively independent of distributed effects, so their approximation in linear lumped models may describe all of the "important" behavior.

- They facilitate efficient teaching through explicit organization. It can be argued that the only systems which are well-understood are those for which a comprehensive and accurate model exists. One of the most difficult aspects of teaching is the distillation of a large, complicated set of facts into a concentrated essence of orthogonal relations. Successful simple models of biological phenomena do just that. Thus, when a simple model is available, it should be used to convey the essential information concisely in an organized framework.

1.4 Model requirements for teaching hemodynamics

Any model to teach hemodynamics should be sophisticated enough to embody the behavior which is being taught to students, but it should not be so complicated that it cannot be understood in and of itself. Although every model will depart from reality to the extent that the assumptions made when constructing it are false, it is important that the model paint a valid picture of the real world. All model assumptions should be presented in a straightforward manner, so that students do not accept the model's version of reality, but maintain a separate understanding of the model and of the real system which the model is meant to describe.

Such a computer simulation of hemodynamics would provide correct results for simulated physiology experiments, but would not be so complicated that it is viewed as an incomprehensible jumble of constituents and relations. Ideally the underlying model would be readily

understandable, so that students using the simulation have a concrete understanding of the principles governing the model, even to the extent of being able to derive the governing equations.[1] Justifiability of model findings is also important.

A computer model of hemodynamics should also encourage real-time problem solving on the part of the student. For a dynamic system such as cardiovascular hemodynamics, the user should have control over the temporal course of the simulation. The time-varying nature of the model should be reflected in a user interface which allows the user to make transient or lasting changes to system parameters, and which tracks the patient over the course of time. One might define the ideal simulation as an instrumented patient, who is in all respects real except for complete control over physiology and measurement ability. While a living, breathing human simulation is unrealistic and unnecessary for the teaching of hemodynamics, one could combine the real-time dynamic nature of the human body and the data available from catheterization with the ultimate control and observability of a computer model to achieve a "virtual patient" for the purposes of hemodynamic study.

[1] This would be even more important if the simulation were used directly in a patient care situation: un-critical reliance on an inappropriate model might lead to inappropriate patient management, while disregarding the results of model study might cause harm to the patient.

Chapter 2

Background

Cardiovascular system modelling has enjoyed popularity through the years as new technology has been available for the exploration of differential equations and as physiologists have increased their ability to quantify physiological parameters. Researchers have attacked the modelling problem at several levels. Some examples of subjects which have been intensely studied include the following: material properties of the components of vascular walls and blood, flow in branching tubes, the distributed impedance of arterial and pulmonary trees, the force-length-velocity relationship in the heart, lumped models of the integrated system, and hemodynamic control systems.

2.1 Previous lumped models of hemodynamics

When analog computers became available for experimentation, interest in lumped models of the cardiovascular system increased. Grodins in 1939 [20] was one of the first to formulate an integrated model of the cardiovascular system in terms of electrical circuit components. His model was developed as a guide for exploring the systemic implications of new concepts in cardiovascular physiology. He began by defining the steady-state operation of the ventricle, and then combined two ventricles with a pulmonary circuit to obtain equations describing the Starling heart-lung preparation. The addition of a systemic circuit then allowed for investigation of the behavior of the closed system. The resulting fourteen-parameter model proved to be a useful

starting point for integrative physiology.[1]

Defares, Osborn, and Hara [15] built an analog computer model in the early 1960's which was stripped of unnecessary complexity and limited in nonlinearities and which produced results intriguingly similar in gross detail to those acquired from humans. They described the model in terms of a six node electrical circuit. Each node consisted of a capacitor to ground connected to two adjacent nodes by impedances and, in the case of the left and right ventricles, diodes. The left and right ventricular capacitances varied to simulate the time-varying elastance model of contraction. The model circuit was realized using operational amplifiers and other analog computer components to produce an instrumented, real-time model of hemodynamics.

Beneken [3] provided an example of a nineteen compartment model which incorporated nonlinearities and ventricular interactions in an attempt to produce a comprehensive short term model of hemodyanmics and baroreceptor reflex control.

In the 1970's, more effort was focused on digital computer models and less on analog computers. This led to great increases in model complexity and thus the numbers of parameters, but a decrease in time resolution for such complex models, because digital computers, though facile with complicated data structures, are slower at solving time-evolving simultaneous equations than are their inherently parallel analog counterparts. Guyton et al. developed refined computer models of the cardiovascular system over longer time scales, and has built an incredibly complex model [25] which embodies, in over 350 equations, the behavior of interacting cardiovascular, renal, and respiratory systems including hormones and blood chemistry.

2.2 Previous cardiovascular teaching with models

Physical hydrodynamic and analog computer models have long been used for laboratory demonstrations of fundamental concepts. Beeuwkes and Braslow pioneered the use of the analog computer for augmenting the hemodynamics teaching laboratory at Harvard Medical School [2]. As digital computer technology acquired the ability to solve systems of equations rapidly, computer simulations continued to augment and replace wet laboratories, dramatically increasing the reproducibility and precision of experiments.

[1] *Integrative physiology* refers to the work of synthesizing a model of the complete organism from constituent models of individual organs.

2.2.1 "MacMan" hemodynamics teaching model

A group of faculty led by Dickinson at MacMaster University developed a digital computer model of the systemic circulation, including a the baroreceptor reflex, in 1971 [17]. This may be the first reported use of a digital computer in medical education. This simulation produced a one minute record of arterial pressure and heart rate values, with cardiac output, mean capillary pressure, right atrial pressure available provided at the end of each simulation. The operator had control of arterial and venous resistance, cardiac contractility, transthoracic pressure, pericardial restriction to filling, and blood volume. In addition, the baroreceptor setpoint could be altered or the reflex system disabled. The simulation was run in batches on an IBM 1132 computer, and could also be run on a PDP-8. In order to stimulate interest and add entertainment value, symptoms appropriate to the current hemodynamic state were printed out with each set of results.

2.2.2 Katz: Windkessel arterial pressure model

Katz et al. were one of the first groups to produce a *real-time* digital computer model for teaching hemodynamics [29]. Although complex real-time analog models existed, a digital computer was preferred for its more flexible human/machine interface over that available with all-analog systems. His computer, programmed in FORTRAN to simulate a simple *windkessel*[2] model of the circulation, produced output on a polygraph recorder via D/A converters in real time. Students were able to investigate different physiologic states represented by variations in heart rate, stroke volume, total peripheral resistance, compliance, and valve competence. In order to allow real-time operation on the available computing machinery (a PDP-12), the mathematical model was kept as simple as possible. Despite its primitive nature, the program developer cited overwhelming student enthusiasm for the program, and instructors were encouraged by the additional hands-on laboratory experience it offered to students.

[2]The term *windkessel*, translated "wind-kettle," refers to a peripheral circulation modelled as a single capacitative compartment which drains through a resistive outflow path, producing an exponential decay of pressure during diastole.

2.2.3 Campbell: canine hemodynamics model

By 1982, it became feasible to digitally simulate a complete, closed cardiovascular system model similar to that of Defares et al. (see Section 2.1, page 22) Campbell et al. [12] developed a five compartment model of canine hemodynamics, incorporating linear vascular elastances and resistances with a time varying ventricular capacitance as proposed by Sunagawa and Sagawa [60]. The resulting set of five first-order differential equations was solved by a fourth-order Runge-Kutta integration method. Interaction was done via a graphics display terminal and pen plotter on a Hewlett-Packard 1000 computer. Due to the complexity of the model and the available computer power, each simulation run required a significant turn around time to obtain results.

2.2.4 Peterson and Armstrong: CVSAD

Building on the work of Campbell et al., Peterson and Armstrong published a program for the IBM PC in 1985 entitled "Cardiovascular Systems and Dynamics" (CVSAD) [47]. Using a similar model to that of Campbell, this program allows for experiments on individual components of the circulation, for example the *windkessel* circulation model and the Starling heart-lung preparation. There is also a model of the closed circulatory system which relates cardiac output to preload and afterload, but which does not solve the complete electrical circuit.

2.3 Previous work in this laboratory

2.3.1 Robert Sah: Cardiovascular Simulator

A lumped parameter six compartment model based on Defares et al. and closely resembling the parallel work of Campbell and Peterson was chosen in 1984 by Sah for teaching hemodynamics [54]. This model represents left and right ventricles, systemic arteries and veins, and pulmonary arteries and veins as linearly capacitative compartments connected by linearly resistive conduits. The model is described in terms of the analogous electrical circuit schematic, where charge represents blood volume, current flow, and voltage pressure (Figure 2-1). The atria are not explicitly modelled: their effect is partially absorbed by choice of constituent parameters for neighboring compartments. Each compartment is defined by a volume at zero pressure, a capacitance, and a real-valued resistance. The inertial effects of blood flow are ignored. Normal values

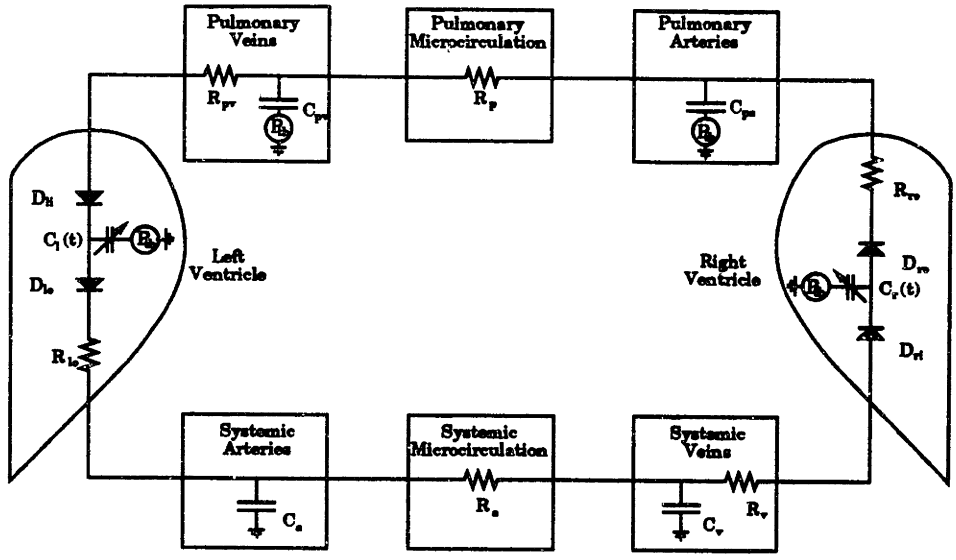


Figure 2-1: Circuit diagram of Sah's lumped parameter hemodynamic model. In this electrical analogy, blood (electrical charge) is pumped counterclockwise in the circuit by time-varying ventricular capacitors.

of many cardiovascular parameters pertinent to compartmental models have been determined, among them total peripheral resistance, vascular elastances, blood volume, cardiac contractility, heart rate, transthoracic pressure (the P_{th} voltage sources in Figure 2-1), and normal ranges for pressures, volumes, and flow rates.

A set of six coupled, linear, time-varying, first order, ordinary differential equations describes the system (see Section 4.2.1, page 45), which was implemented using a modified Runge-Kutta integration method (see Section 4.2.3, page 47). Its time-varying nature has several sources:

1. The cardiac pump is modelled as a time-varying capacitance. Low capacitance corresponds to the systolic contraction phase of the cardiac cycle; high capacitance corresponds to the diastolic filling phase of the cycle.

2. The heart inflow and outflow valves as modelled allow unrestricted but unidirectional blood flow; therefore, they may be viewed as time-varying resistances or as a time-varying circuit topology rather than as nonlinear elements.

3. In addition, any intervention by a control model or student-directed parameter change would

cause other constituent parameters to vary.

Sah built a user interface for the simulation which provided for parameter display and modification, calculation of a data set for a specified number of heart beats, and plotting of any hemodynamic quantities over the course of the data set. Plots were made on a Tektronix 4014 graphics terminal emulator and printed through the screen-dump capabilities of an attached dot-matrix printer. The user interacted with the system by choosing numbered items to navigate through layers of menus. There were sets of parameters corresponding to the normal state and several predefined hemodynamic compromises, such as blood loss, primary hypertension, myocardial infarction, and aortic stenosis. In addition, several clinical case simulations had been developed, each with a case history text and an accompanying set of parameters. Students in the Harvard-M. I. T. Division of Health Sciences and Technology used this simulation to investigate hemodynamic principles, and were assigned to present a report based on simulation of a particular class of pathological processes or based on a particular patient case simulation. Although some students expressed reservations about spending time on mathematical models rather than concentrating exclusively on descriptive clinical knowledge, most were enthusiastic about the concept of cardiovascular simulation. However, even the eager students complained about the excessive time required to learn to use the program.

2.3.2 Chueh and Bhatty Simulator

Apurva Bhatty and Henry Chueh converted Sah's program (see Section 2.3.1, below) to run on the IBM PC for use by students in the Harvard Medical School's New Pathway Program [8]. Though the simulation runs slowly, it has all the capabilities of Sah's original program.

Chapter 3

Problem Statement

Although Sah's original cardiovascular simulation was a success in the classroom, it contained several weaknesses:

- It had not been experimentally verified, though data from the simulation appeared to behave qualitatively similarly to the cardiovascular system.
- The simulation lacked any control system model, whereas the physiological system is tightly controlled over the short term primarily by an arterial baroreceptor reflex system.
- The original user interface proved cumbersome for students, which limited student-simulation interaction.

In order to improve on previous efforts, I proposed to accomplish the following:

1. Verify that the updated hemodynamic model is a valid description of human physiology, in that
    a. all of its parameters have solid experimental bases, and
    b. the state variables and derived variables take on appropriate values as compared with published norms.
2. Design and implement a baroreceptor reflex model using a moving-average proportional controller, with delay and gain parameters established independently from the literature.

Validate the model by comparing to physiological data the simulator's response to simulated step changes in parameters.

3. Improve the user interface to the simulator by implementing user-friendly graphics-oriented techniques based on the X11 Window System and the Motif window manager now supported by Project Athena at M. I. T.

4. Evaluate the teaching effectiveness of the new version of the simulator (CVSIM) in the context of M. I. T. subjects *HST–090, Cardiovascular Pathophysiology* and *6.022J/2.792J/-HST–542J Quantitative Physiology: Organ Transport Systems.*

Chapter 4

Verification of the Hemodynamic Model

When providing a simulation to students with the purpose of refining and exercising their knowledge of physiology, it is important to have confidence in the validity of the model; this confidence can be gained through an understanding of the inherent assumptions and the physiological bases for parameter choices. I will begin by describing necessary assumptions for the circuit topology chosen by Sah (see Section 2.3.1, page 24), and then list physiological experiments from the literature to support the choice for each parameter value.

4.1 Model design

The systemic and pulmonary circulations have a distributed, complex branching geometry. The walls of vessels contain elastic fibers which provide a linear elastance, or capacitance, but there is also smooth muscle and collagen which add adaptive and nonlinear behavior. In addition, blood behaves as a non-Newtonian fluid, particularly in the small vessels. Only by making certain assumptions about the behavior of these nonlinear material properties under specific circumstances can we hope to describe the system comprising them in a coherent manner.

4.1.1 Assumptions for circuit topology

Lumped parameters

The primary assumption of this model is that the cardiovascular system can be treated as a collection of lumped elements. Whether this is a valid approximation depends on the questions to be addressed. For example, if we wished to understand the shape of the pulse waveform in the peripheral arteries, which depends on the distance along the artery, the transmission impedances, reflections of previous pulses, and other factors, a lumped model clearly would not be appropriate. On the other hand, if we wished to understand the "average" behavior of pressures in the system, for instance the peak and minimum pressures and the first order time constants, a lumped model might well be adequate. In the case of arterial pressure, lumping the systemic arteries and systemic resistances into one element apiece will remove any possibility of modelling distributed effects, but may model the pressure at the root of the aorta quite nicely, because the pressure waveform there is dominated by the locally linear arterial elastance and the driving function of the heart.

Linear elements

The second major assumption is that the lumped elements can be treated as linear. The capacitance of each compartment can be treated as linear only over a range of pressures in which the measured pressure–volume relation is approximately linear. This range corresponds roughly to the range in which the same elastic fibers in the vascular wall are stressed. Beyond this limit, low-strain collagen fibers become stretched and the elastance of the wall is dramatically reduced.[1]

When a vessel's transluminal pressure is sufficiently negative, it will collapse. Since it is impossible for volume to be negative, this represents another nonlinearity, which can become important for the venous side of the circulation. Therefore, as long as transluminal pressures remain between zero and the linear elastic limit, compartmental capacitances can be adequately modelled as linear capacitors.

The systemic arteries unfortunately have a nonlinear elasticity over the physiological range,

[1]The linearity of each compartment must be investigated individually, as physical characteristics vary greatly between the heart, arteries, veins, and pulmonary vessels.

due to the presence of multiple fiber types and tensions. However, for simplicity, arterial capacitance is treated as linear in this model (see Section 4.1.3, page 34). Also, the pulmonary vascular resistance is composed of parallel collapsing tubes at varying external pressures. A desirable future enhancement to the model would be to add nonlinearity at least to the systemic capacitance and pulmonary resistance.

Viscous-dominated flow

The inertial effect of blood acceleration is neglected in the model. It is well known that in the high-pressure systemic side of the circulation, inertial energy is small relative to the amount of work dissipated by viscous effects. However, in the pulmonary system and within the heart, acceleration can play a significant role.

For instance, the deceleration of blood just after it has entered the aorta by ventricular ejection and its recoil against the closing aortic valve causes a small glitch in the aortic pressure tracing, called the dicrotic notch. The absence of an inductance, to model inertance in the systemic arteries, precludes the ability to reproduce the dicrotic notch.

In the pulmonary circulation, pressures are more likely to be influenced by inertia because of the much lower pressures and resistances. The inertial (kinetic) energy in the pulmonary tree is estimated at five percent of the total external work [15], which is an acceptably low level for the purpose of this simulation.

Insignificance of atrial contraction

The proposed network lacks atria. In man, the atria pack the ventricles with blood in late diastole, so that stroke volume is maximized, and they increase the capacitance for venous return during systole. The atria have relatively little physiological impact during normal resting conditions, but during exercise, atrial contraction is critical [6]. This is because at resting heart rates, the ventricle is almost completely filled before the atrial contraction, whereas at high rates, systole can impinge on the rapid filling phase of diastole, and the atrial contraction is necessary for adequate ventricular filling. Thus, the absence of atria is not important at low heart rates. At higher heart rates, the reduction in ventricular filling can be counteracted by manually increasing the ventricular diastolic capacitance or decreasing the inflow resistance.

Anatomic simplicity

There is no interaction between topologically nonadjacent compartments. For instance, the left and right ventricles share a common wall and in many respects are a single organ. However, the pressure in the left ventricle does not contribute significantly to right ventricular hemodynamics. Arteries and veins also are usually physically adjacent, and venous pressure tracings thereby reflect the pulsations in the adjacent artery. Such adjacency effects, as well as the normally observed noise and transducer inaccuracies, and the effects of external systems such as the variation in pulmonary parameters with breathing, are absent from this model.

Closed system

The modelled cardiovascular system is closed. Fluid transport between the vascular and extravascular spaces is ignored, since it is not physiologically relevant at normal pressures which do not produce edema.

4.1.2 Origins of parameter values

Each compartment is defined by a set of three parameters:

1. a linear capacitance,

2. a zero-pressure filling volume, which is the volume remaining when the pressure acting on the capacitance is zero, and

3. an external pressure opposing the internal pressure felt by the compartment.

These compartments are connected by conduits defined as linear resistors. Most of the parameters can be easily derived from a measured pressure–flow ($P-Q$) characteristic across the constituent of interest. For a resistance between compartments i and j,

$$R_{ij} = \frac{P_i - P_j}{q_{ij}}$$

The capacitance of a compartment can be obtained from the slope of the pressure – volume ($P - V$) relation for that compartment:

$$C_i = \frac{dV_i}{dP_i}$$

Often in the literature, the $P - V$ curve is approximated by a line through the origin and C_i is defined simply as

$$C_i = \frac{V_i}{P_i}$$

The zero-pressure filling volume for each compartment is adjusted to make its sum with capacitative volume equal to measures of normal blood volume for that compartment. These parameters take on values which depend on the size and physiological state of the individual. For the purpose of defining a reasonable model, it is necessary only to develop a set of compatible parameters, each within the normal range for a 70 kilogram individual, which also give rise to appropriate RC time constants for each node in the system.

Next, I will discuss parameter choices for each compartment.

4.1.3 Systemic circulation parameters

The systemic circulation is anatomically divided into thick-walled arteries which carry blood from the heart and maintain high pressure though elastic recoil and muscular strength; arterioles, capillaries, and venules which together make up the bulk of the resistance to blood flow; and thin-walled veins which are usually partially collapsed and direct blood back to the heart at low pressure. This system is modelled as two capacitative compartments in series with a resistance.

Arteriolar resistance, R_a

Systemic resistance is perhaps the most easily produced value of all the compartmental parameters.[2] One needs only to measure central venous and arterial pressures, subtract, and divide by cardiac output. Average arterial pressure in a healthy resting male ranges from 90 to 100 mm Hg, with an average about 96 mm Hg, and average central venous pressure of approximately

[2] Systemic resistance is also one of the most variable parameters within an individual; for example, during exercise, R_a can decrease by a factor of three [6] as blood flow to skeletal muscle is increased.

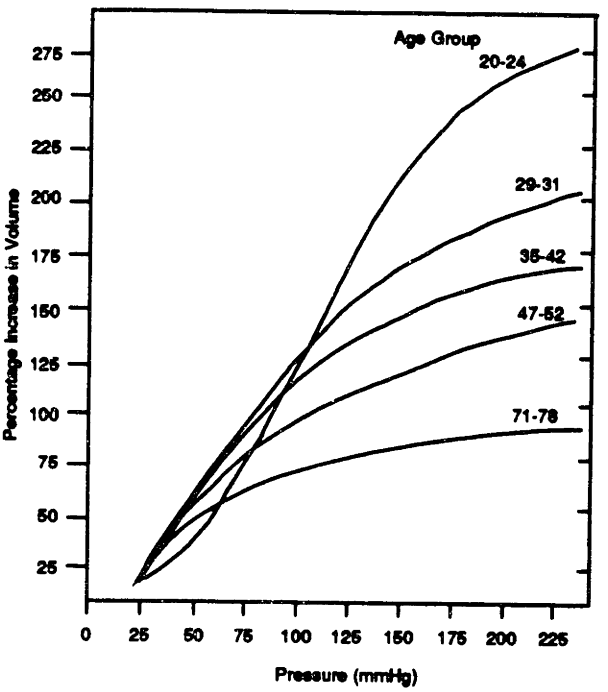


Figure 4-1: Changes in arterial capacitance with age. Data from aortas obtained at autopsy. (Redrawn from Hallock and Benson [28], 1937.)

6 mm Hg. Since cardiac output in an individual at rest with these pressures is normally about 90 ml/s, this corresponds to a total peripheral resistance of $(96 - 6)/90 = 1$ mm Hg s/ml, by Ohm's Law. For this reason, 1 mm Hg s/ml is renamed as the Peripheral Resistance Unit (PRU).

Arterial capacitance, C_a

Many estimates have been made of the lumped capacitance of the arterial tree. Defares et al. [15] used a value of 1.9 ml/mm Hg, and the from dog studies by Sato et al. extrapolates to 2.6 ml/mm Hg [55]. However, the pressure-volume curve is not linear, but sigmoid, and changes dramatically with age (see Figure 4-1); therefore, these numbers do not have a great deal of meaning for linear modelling purposes. It is important, however, to maintain an appropriate time constant for flow through the systemic circulation. The exponential time constant for flow from systemic arteries to veins, $(RC)_a$, has been measured at 1.9 s [55]. Sah's chosen value of 1.6 ml/mm Hg, after Beneken [3], gives $(RC)_a = R_a \frac{C_a C_v}{C_a + C_v} = 1.57$ s for the arterial time constant,

which is near the published findings.

Venous resistance, R_v

Compared to the systemic microvasculature, resistance to blood flow in the veins is very low. When pressure in a peripheral vein is compared to central venous pressure, taking care to remove differences in transducer height, very little average pressure difference is detected. In a more complex model [3], Beneken used values for individual peripheral venous resistances which when taken together produce an overall venous resistance of 0.080 PRU. Sah's choice of 0.050 is near this value. Consideration of the ventricular filling time constant might lead to an even lower value. This will be addressed in the following section.

Venous capacitance, C_v

The lumped venous capacitance is difficult to estimate, because the precise percentage of blood in distending versus nondistending volume is unknown. Defares et al. estimated C_v at 100 ml/mm Hg [15]. A similar value was found by Sato et al., who reported 92 ml/mm Hg [55].

As in the case of the arterial tree, the venous capacitance should lead to an appropriate time constant for venous drainage. The lack of atria in the model complicates the choice of reasonable values for C_v and R_v. The normal rapid filling time for the right ventricle has been measured at 0.0–0.4 s [10]. If this "rapid filling" corresponds roughly to two time constants of exponential decay, then $(RC)_v = R_v \frac{C_v C_{r,dias}}{C_v + C_{r,dias}}$ should be less than 0.2 s. Sah's chosen values of 0.05 PRU for R_v, 100 ml/mm Hg for C_v, and 20 ml/mm Hg for $C_{r,dias}$ (see Section 4.1.5, page 36) give an $(RC)_v$ of 0.833 s, which is clearly higher than desired. Part of this discrepancy may be explained by physiology: the ventricles are fairly stiff-walled structures, and when they expand after systole, the walls act to suck blood into the ventricle from the atria and veins. This suction effect may account for the more rapid observed rapid filling time than the model can produce.

Because these chosen parameters lead to a longer simulated filling time, and because of the absence of atria, heart rate increases in the model will not increase cardiac output as much as expected from a knowledge of physiology (see Section 4.1.1, page 31).

4.1.4 Pulmonary circulation parameters

The pulmonary vasculature has the comparatively easy task of forcing blood through the low-resistance lung vasculature and returning it to the left heart. The pulmonary system is therefore a low pressure system.

Unfortunately, an accurate model of the pulmonary system would require a nonlinear "vascular waterfall" resistance and possibly a nonlinear pulmonary artery capacitance. The linear model presented here is intended to produce acceptable behavior for the integrated system, and is only generally indicative of pulmonary hemodynamics.

Pulmonary vascular capacitances, C_{pa} and C_{pv}

Sah chose Beneken's values [3] of 4.3 ml/mm Hg for pulmonary arterial capacitance and 8.4 ml/mm Hg for pulmonary venous capacitance. The value for C_{pv} includes the capacitance associated with the left atrium.

Pulmonary vascular resistance, R_p

The pulmonary resistance has been measured at 0.08 PRU [38]. This reported value, however, belies the fact that the pulmonary resistance is nonlinear. Pulmonary resistance decreases as cardiac output increases, because of the recruitment of additional parallel capillary beds higher in the lungs. If we keep pulmonary arterial pressure relatively constant, however, we should still be able to model the peripheral circulation adequately with a linear pulmonary vascular resistance.

4.1.5 Cardiac parameters

Diastolic compliance, $C_{l,dias}$ and $C_{r,dias}$

Ventricular capacitance during diastole can be measured by varying volume and measuring the pressure change. Although the thick, muscular walls of the heart do not resemble the elastic tubes of the vascular tree, the measured capacitance is linear over most of the normal range, until the ventricle becomes filled near capacity. Mirsky [41] compiled left ventricle end-diastolic dP/dV for eight patients: values ranged from 0.15 to 0.6 mm Hg/ml. If translated directly to $C_{l,dias}$ as dV/dP, this would suggest a value between 1.6 and 6.7 ml/mm Hg. However, this is the slope

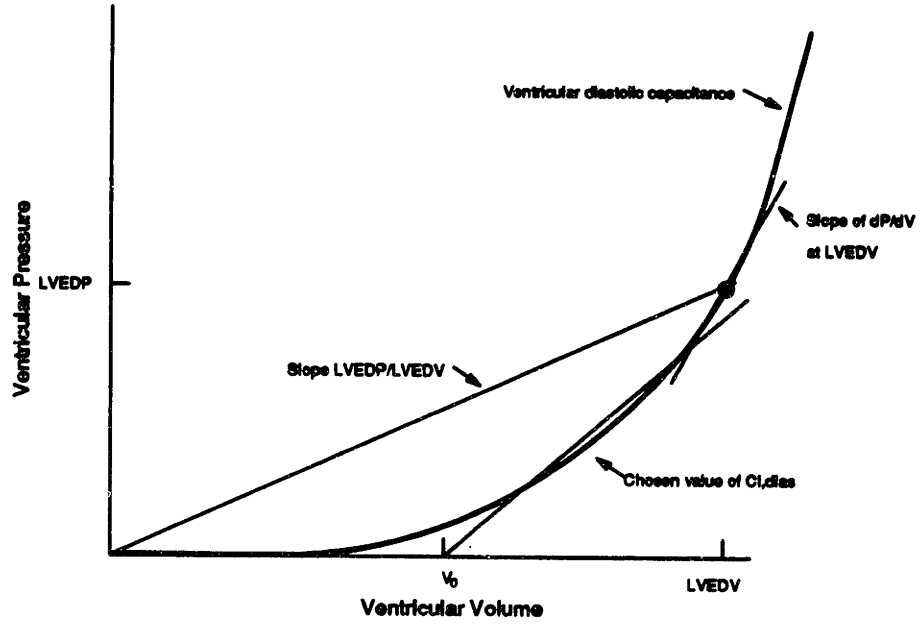


Figure 4-2: Compartmental capacitance choice. The ventricular diastolic capacitances are reduced dramatically as they fill to maximum volume. It is therefore not reasonable to use the end-diastolic dV/dP, but rather some slope which approximates the most active portion of the curve (between 5 and 30 mm Hg) offset from the origin by the zero-pressure volume.

measured at the most stiff point on the curve, as the ventricle is becoming stretched to capacity. A reasonable approximation would be to choose a value which lies between this measured late diastolic capacitance and end-diastolic V/P, which is approximately 12 ml/mm Hg (Figure 4-2). Sah's value of 10 ml/mm Hg for $C_{l,dias}$ falls between these extremes.

The right ventricular diastolic capacitance is known to be approximately double that of the left [16], therefore Sah's value of 20 ml/mm Hg seems reasonable.[3]

Minimum systolic capacitance, $C_{l,sys}$ and $C_{r,sys}$

Minimum systolic capacitance, or maximum elastance ($E_{max} = 1/C_{sys}$) has been considered as an index of ventricular function. A normal value of 4 mm Hg/ml for E_{max} of the left ventricle has been suggested by Grossman [21]. Sah's model uses 0.4 ml/mm Hg for minimum left ventricle capacitance, and 1.2 ml/mm Hg for minimum right ventricle capacitance. Although this value for

[3]Note that these linear diastolic compliances are unrealistic for end-diastole at slow heart rates, because elastance should be reduced dramatically as the heart fills to capacity.

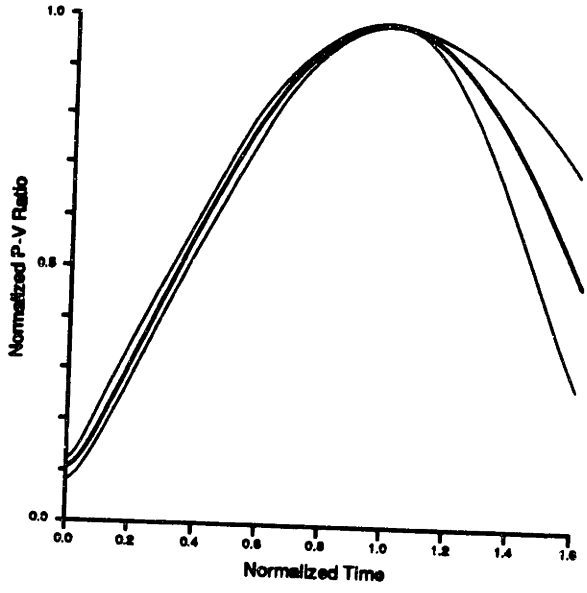


Figure 4-3: Idealized average left ventricular elastance curve in the dog, redrawn from Suga and Sagawa [59]. This curve, which is normalized for elastance from 0 to 1, was used to obtain the systolic capacitance curve as a function of time for both left and right ventricles, as capacitance varies between the chosen C_{dias} and C_{sys}.

$C_{l,sys}$ is less contractile than the suggested value of 1/4, it is reasonable, as will be demonstrated by the generated arterial pressure.

The right ventricular capacitance was chosen in light of the fact that the right ventricle should generate pressures approximately 1/3 of those generated by the left ventricle. In addition, this value falls within the range of 0.35 and 1.6 for $C_{r,sys}$ measured in humans by Dell'Italia and Walsh [16].

Time-varying capacitance, $C_l(t)$ and $C_r(t)$

The ventricular capacitances vary between $C_{l,sys}$ and $C_{l,dias}$, and between $C_{r,sys}$ and $C_{r,dias}$ through a predetermined function in time for each ventricle. Our time-varying elastance curves are taken from idealized canine elastance curves (Figure 4-3). The curve shown is from averaged dog data; its shape can be used as a model for the human heart because of the similarities of arterial pressures and compliances between dogs and humans. However, the averaging process may have masked a more gently-sloping earlier portion of the curve, as is indicated from sample

$E(t)$ tracings from the same paper [59]. The $E(t) = 1/C(t)$ currently used is one half cycle of inverted cosine scaled in time to occupy 2/3 of the time for systole, followed by 1/2 cycle of upright cosine in the final 1/3 of systole. This composite curve is translated and scaled to range between $1/C_{dias}$ and $1/C_{sys}$ for each ventricle.

The $C_l(t)$ and $C_r(t)$ curves can be fitted more precisely to the human by ensuring that their maximum slopes fit known human data. One approach is to use an isovolumic measure of contractility such as the Contractility Index $((1/P)\,dP/dt)_{max}$, [46], [60], [41], [7], and to compare its normal value to the model. For our simulation,

$$\frac{1}{P_l}\frac{dP_l}{dt} = -\frac{1}{C_l(t)}\frac{dC_l}{dt}$$

during isovolumic contraction.[4] Closed-form calculation for our model gives

$$\left(-\frac{1}{C_l(t)}\frac{dC_l}{dt}\right)_{max} = 28\ \mathrm{s}^{-1}$$

for the described $C(t)$, at a heart rate of 72 beats per minute. This is lower than a reported normal range [46] of $44.0 \pm 1.8\ \mathrm{s}^{-1}$, and in fact is closer to that study's range of abnormal patients, $28.1 \pm 1.7\ \mathrm{s}^{-1}$. This leads to a conclusion that the current shape of $E(t)$ is too gentle during early systole, and should be replaced by function with a more sharply sloping early systole.

Several investigators use the top half of a sinusoid for their $E(t)$; however, this would lead to a contractility index of $280\ \mathrm{s}^{-1}$ for the given preload, which is maximum at the very onset of systole. In a more sophisticated model of the ventricle, Beyar and Sideman [7] used such a half-sine $E(t)$; it is uncertain how they arrived at their value of $((1/P)\,dP/dt)_{max} = 55\ \mathrm{s}^{-1}$.

Because the current $E(t)$ provides near-physiological values compared to the standard half-sine model from the literature, I have maintained the current $E(t)$; however it must be understood that this model reflects a relatively weak isovolumic contractility. An $E(t)$ determined from a

[4] This can be seen by differentiating the pressure-volume relation for a linearly capacitative compartment, because during isovolumic contraction,

$$\begin{aligned} V &= C\,P + V_0 \\ \frac{dV}{dt} &= C\frac{dP}{dt} + P\frac{dC}{dt} = 0 \end{aligned}$$

multiple-parameter fit would be an improvement on this provisional model.

Valvular resistances, R_{ro}, R_{li}, and R_{lo}

The right heart inflow resistance resulting from passage through the venous and tricuspid valves is very small. It is much less that of the neighboring systemic venous resistance, and therefore was incorporated into R_v.

The right heart outflow resistance has been estimated at 0.003 PRU [52], [15]. This value is used for R_{ro} in Sah's model.

The left heart inflow resistance is a determinant of left ventricular filling time. It has been estimated to be 0.009 PRU [15], and extrapolation from Sato et al. [55] gives 0.016 PRU. Sah's choice of 0.01 PRU is congruent with these other estimates. These choices for R_{li} give an inflow time constant of $R_{li}\frac{C_{pv}C_{l,dias}}{C_{pv}+C_{l,dias}} = 0.05$ s. This agrees with the observation of rapid filling time in the left ventricle between 0.04 and 0.08 s [10].

The left heart outflow resistance was estimated by Defares at 0.01 PRU, and by Sah at 0.006 PRU. It can be determined more precisely for a given case by measuring the LV-Aorta pressure gradient. Any pressure gradient less than about 0.5 mm Hg is within normal measurable physiological limits. Left ventricular outflow is affected little by such a small resistance in the normal case, since the time constant for ejection across this resistance at peak systole is $R_{lo}\frac{C_{l,sys}C_a}{C_{l,sys}+C_a} = 0.002$ s. Output from the left ventricle is determined much more by ventricular filling, contractile state, and arterial pressure. Choice of a lower outflow resistance leads to a very short ejection time constant, and as a result the set of equations for the closed system becomes more stiff, requiring more effort to solve numerically (see Section 5.3.1, page 61).

4.1.6 Blood Volumes

The average volume of blood in each compartment of the cardiovascular system depends on the physiological state, position against gravity, and anatomical stresses. However, some general idea of normal volumes is necessary in order to present the student with reasonable values.

Blood volumes have been tabulated in Guyton's and Mountcastle's physiology texts [24], [39]. approximately 15% of the blood volume is to be found in the aorta, systemic arteries and arterioles, 69% in the capillaries, venules, and systemic veins, 9% in pulmonary circulation, and

7% in the heart. I will compare these percentages directly with our model.

We will begin with the total blood volume, and then divide this into the various compartments at standard pressures to obtain each compartment's zero-pressure filling volume.

Total blood volume, V_{tot}

Blood volume in the human varies over a wide range, from less than 3 liters in some small women to over 6 liters in some large, athletic men. Athletes, especially runners, have an expanded blood volume compared to non-athletes. Blood comprises approximately 77 ml/kg body weight in the average male, and 67 ml/kg body weight in the female [57]. Considering these figures, Sah's choice of 5 liters is an appropriate blood volume for a standard 70 kg male.

Systemic volumes, V_{a0} and V_{v0}

Arterial volume At 90 mm Hg, the arterial capacitance has 1.6 ml/mm Hg $\times$ 90 mm Hg = 144 ml of stressed volume. Sah's value for zero-pressure volume was 715 ml, bringing the total arterial volume to 859 ml, or 17% of the total volume. This is fairly close to the anticipated 15%.

Venous volume At 6 mm Hg, the peripheral veins have $6 \times 100 = 600$ ml of stressed volume. Adding Sah's unstressed volume of 2500 ml brings the total average venous volume to 3100 ml, or 62% of the total blood volume Sah's systemic venous volume is therefore near the estimate of 69%.

Pulmonary volumes, V_{pa0} and V_{pv0}

In spite of the percentage values given above, there is a wide range of reported pulmonary volumes. For instance, Burton reported 8% of the blood volume in the pulmonary arteries, and 18% in the pulmonary veins, respectively, in a young man [11]. If this is divided relatively evenly between arteries and veins as suggested by Guyton [23], Sah's values of $V_{pa0} = 90$ ml and $V_{pv0} = 490$ ml for unstressed volumes are appropriate, because when stressed to normal pressures this gives approximately 550 ml for V_{pa} and 450 ml for V_{pv}.

Compartment	R_i (PRU)	R_o (PRU)	C (ml/mmHg)	V_0 (ml)
Left Heart	0.01	0.006	0.4—10	15.
Systemic Arteries	0.006	1.00	1.6	715.
Systemic Veins	1.00	0.05	100.0	2500.
Right Heart	0.05	0.003	1.2—20	15.
Pulmonary Arteries	0.003	0.08	4.3	90.
Pulmonary Veins	0.08	0.01	8.4	490.
Systemic Parameters:	Total blood volume		5000 ml	
	Heart rate		72 beats/min	
	Transthoracic pressure		–4 mm Hg	

Table 4.1: Summary of simulation parameters. R_i and R_o are inflow and outflow resistances in peripheral resistance units (mm Hg s/ml), C is capacitance, and V_0 is zero-pressure filling volume. Note that inflow resistance for each compartment is identical to the outflow resistance for the preceding adjacent compartment.

Heart chamber volumes, V_{l0} and V_{r0}

The filling volume of the left ventricle in the human varies from 70 to 100 ml per square meter of body surface area, and since the right ventricle in the steady state must pump the same volume, it must have a similar filling volume exclusive of dead volume. If our average 70 kg man has a body surface area of 1.5 m^2, this corresponds to approximately 100–150 ml for each ventricle at end-diastole. At an unstressed volume of 15 ml for both ventricles, an RV filling pressure of 5 mm Hg and a left ventricular (LV) filling pressure of 10 mm Hg, Sah's values give an RV volume of $C_{r,dias}P_r + V_{r0} = 115$ ml and similarly an LV volume of 115 ml.

4.1.7 Parameter summary

The cardiovascular system parameters described above, with the addition of heart rate and transthoracic pressure, form a complete six-compartment linear model of the uncontrolled circulation. Table 4.1 summarizes these parameters.

These parameter choices directly determine the time constants for flow between adjacent

Location	$\tau = (RC)_{ij}$ (s)
Left heart inflow	0.046
Left heart outflow	0.0019
Arteriolar flow	1.6
Right heart inflow	0.83
Right heart outflow	0.0028
Intra-pulmonary flow	0.23

Table 4.2: Time constants for flow between neighboring compartments in the uncontrolled simulation. The heart outflow time constants were calculated using the minimum systolic capacitances, whereas the inflow time constants use the diastolic heart capacitances.

parameters, by the relation $(RC)_{ij} = R_{ij}\frac{C_i C_j}{C_i + C_j}$ where R_{ij} connects compartments i and j. The adjacent-compartment time constants are listed in Table 4.2.

4.2 Model implementation

Implementation of this model on a digital computer requires the formulation of the set of equations defining the system, assignment of initial values to all state variables, and an integration method to estimate the evolution of the state variables through time. In addition, since this is a real-time dynamic simulation, parameter update constraints are required to permit intra-simulation parameter modifications.

4.2.1 Governing equations for model of hemodynamics

State equations from KCL analog

It is a matter of elementary network theory to write the equation analogous to Kirchoff's Current Law for electrical circuits for each node in the network. Since this circuit has a simple ring topology each equation takes the same form: current into the node from upstream, minus the rate of storage of charge on the capacitor, equals current out of the node. Current into and out of each node is determined by the pressure difference across the resistance between nodes. The

current q_{ij} between adjacent compartments is thus

$$
\begin{aligned}
q_{li} &= \begin{cases} (P_{pv} - P_l)/R_{li} & \text{if } P_{pv} > P_l \\ 0 & \text{otherwise} \end{cases} \\
q_{lo} &= \begin{cases} (P_l - P_a)/R_{lo} & \text{if } P_l > P_a \\ 0 & \text{otherwise} \end{cases} \\
q_a &= (P_a - P_v)/R_a \\
q_{ri} &= \begin{cases} (P_v - P_r)/R_{ri} & \text{if } P_v > P_r \\ 0 & \text{otherwise} \end{cases} \\
q_{ro} &= \begin{cases} (P_r - P_{pa})/R_{ro} & \text{if } P_r > P_{pa} \\ 0 & \text{otherwise} \end{cases} \\
q_{pv} &= (P_{pa} - P_{pv})/R_{pv}
\end{aligned}
$$

The state form of the node equations can be written in terms of these q_i's.

$$
\begin{aligned}
\frac{dP_l}{dt} &= \frac{q_{li} - q_{lo} - (P_l - P_{th})\, dC_l(t)/dt}{C_l(t)} \\
\frac{dP_a}{dt} &= \frac{q_{lo} - q_a}{C_a} \\
\frac{dP_v}{dt} &= \frac{q_a - q_{ri}}{C_v} \\
\frac{dP_r}{dt} &= \frac{q_{ri} - q_{ro} - (P_r - P_{th})\, dC_r(t)/dt}{C_r(t)} \\
\frac{dP_{pa}}{dt} &= \frac{q_{ro} - q_{pv}}{C_{pa}} \\
\frac{dP_{pv}}{dt} &= \frac{q_{pv} - q_{li}}{C_{pv}}
\end{aligned}
$$

These are the equations which are used in the numerical solution. Starting with an estimated set of pressures, the equations give the local slope, which is used iteratively to estimate the next set of pressures in time.

Matrix form for system

It is apparent that this network produces a linear time-varying first-order differential equation. Its system matrix is zero everywhere except for the main diagonal, the elements adjacent to the main diagonal, and the opposite corner elements.

$$\frac{d\mathbf{p}}{dt} = \mathbf{A}\mathbf{p} + \mathbf{b}$$

Here p is the vector of compartmental pressures, A represents the time constants for exchange between compartments, and b is the "input" to the system via the external pressure offset P_{th} for the thoracic compartments. Note that in our system, these inputs are constant and the A matrix varies as the variation in C_l and C_r provides energy to the system, and as the heart valves regulate the direction of flow.

4.2.2 Initial conditions

The pressures in each compartment depend on the history of the system. In order to compute a solution to plot pressures through time, we must start a set of initial values for pressure.

The pulmonary vessels, heart, and a fraction of the arterial tree volume are located in the thorax; this area of the body is subject to pressurization and depressurization by the diaphragm, whereas the peripheral tissues, especially the leg veins and viscera which store the bulk of the venous blood, are outside this area. Transthoracic pressure is normally about -4 mm Hg during rest, and even lower during inhalation, due to the elasticity of the lungs pulling against a fixed rib cage. Contraction of the abdominal musculature or lower extremities and postural changes can have a marked effect on venous pressure and resistance. For this study, only the transthoracic pressure input is considered, affecting the blood volumes in the heart, pulmonary circulation, and approximately one third of the systemic arteries [31]. All other body compartments are considered to be at atmospheric pressure.

Initial pressures are estimated by a linear algebraic solution developed by Sah. A DC circuit is solved under the assumption that all state variables are constant. Each ventricle is treated twice for the purposes of the DC approximation: the diastolic capacitance is used for the ventricular inflow calculation, and the systolic capacitance is used in the outflow calculation. A system of

eight linear equations, depicting conservation of flow rate, in eight unknowns[5] is solved using a modified Gauss-Jordan reduction to obtain the six initial pressures for a specific set of parameters.

$$
\begin{aligned}
& C_{l,dias}\left(P_{l,dias}-P_{th}\right)-C_{l,sys}\left(P_{l,sys}-P_{th}\right) \\
&= C_{r,dias}\left(P_{r,dias}-P_{th}\right)-C_{r,sys}\left(P_{r,sys}-P_{th}\right) && (4.1)\\
&= T_{sys}\frac{P_{l,sys}-P_a}{R_{lo}} && (4.2)\\
&= T_{tot}\frac{P_a-P_v}{R_a} && (4.3)\\
&= T_{dias}\frac{P_v-P_{r,dias}}{R_v} && (4.4)\\
&= T_{sys}\frac{P_{r,sys}-P_{pa}}{R_{ro}} && (4.5)\\
&= T_{tot}\frac{P_{pa}-P_{pv}}{R_{pv}} && (4.6)\\
&= T_{dias}\frac{P_{pv}-P_{l,dias}}{R_{li}} && (4.7)\\
V_{tot}-V_{0,tot} &= C_{l,dias}\left(P_{l,dias}-P_{th}\right)+C_a\left(P_a-\frac{1}{3}P_{th}\right)+C_vP_v \\
&\quad + C_r\left(P_r-P_{th}\right)+C_{pa}\left(P_{pa}-P_{th}\right)+C_{pv}\left(P_{pv}-P_{th}\right) && (4.8)
\end{aligned}
$$

Eq. (4.1) asserts that the difference between diastolic and systolic volumes are equal in the left and right ventricles. Eqs. (4.2) through (4.7) equate that volume with the volume which flows through each resistance during the time that it is active. For instance, Eq. (4.2) states that the average flow across R_{lo} over one systolic ejection period T_{sys} is also equal to the stroke volume. Eq. (4.8) equates the total stressed blood volume, which is the total blood volume minus all of the compartmental the zero-pressure (unstressed) volumes, with the sum of compartmental stressed volumes for the given pressures.

This set of equations is solved for the eight P_i, and the diastolic values are used as initial values for the integration.

[5] The unknowns are the compartmental pressures: four average pressures, two systolic ventricle pressures, and two diastolic ventricle pressures.

4.2.3 Integration method

The most direct integration method is a fourth-order Runge-Kutta iterative solution as used by Sah, with code taken directly from *Numerical Recipes in C* [49]. For stability, the step size must be chosen to be on the order of the shortest time constant in the system. Unfortunately, the two ventricular outflow paths have very short time constants of 0.002 s. Because ventricular capacitance changes smoothly to force blood from the ventricles, a time step slightly longer than this is allowable; however, time steps significantly over 0.006 s can lead to convergence failure.

An adaptive step size Runge-Kutta method [48] addresses the problem of varying time scales by changing the time step based on the difference between the two limbs of a fifth-order Runge-Kutta step. With this method, the step size is tested and possibly increased or decreased each step by a factor of $\epsilon^{1/5}$, where ϵ is the estimated fractional error in the current step. With this method, the integration sweeps broadly and quickly over the smooth terrain, in this case during diastole, and takes smaller steps through the rough terrain, in our simulation during systolic ejection. In order to maintain visual plot smoothness during diastole, a maximum time step of 0.02 s was introduced.

4.2.4 Parameter update constraints

In order to allow dynamic parameter modification, a set of constraints on instantaneous parameter changes was developed which enforces the conservation of blood volume in each compartment, unless blood volume itself is being modified (Table 4.3). When a parameter is to be modified, a pressure change in the relevant compartment is computed which corresponds to constant compartment volume during a step change in the parameter. These constraints allow dynamic external modifications and control system-directed modifications of the various constituent parameters of the original model.

4.3 Comparison of simulation results to human data

If the model framework and parameter values form a valid representation of physiology, then the average pressures, volumes and flows measured and the general pulsatile waveforms generated should be similar to those obtained from human subjects under similar conditions.

Parameter Type	State variables updated
Capacitance (C_i)	$P_i \leftarrow P_i \frac{C_{i,old}}{C_{i,new}}$
Zero-pressure volume (V_{0i})	$P_i \leftarrow P_i + \frac{V_{0i,old} - V_{0i,new}}{C_i}$
Resistance (R_{ij})	No state variable change.
Total blood volume (V_{tot})	$P_v \leftarrow P_v + \frac{V_{tot,new} - V_{tot,old}}{C_v}$
Transthoracic pressure (P_{th})	For all thoracic compartments, $P_i \leftarrow P_i + (P_{th,new} - P_{th,old})$

Table 4.3: Parameter update constraints required for constant-volume assumption. An instantaneous change of any compartmental parameter will cause a pressure change in that compartment to reflect the new state. As the state variables of the simulation, the pressures in turn determine the compartmental volumes and flows, maintaining a consistent physiological state through the parameter modification.

4.3.1 Average pressure and flow

Table 4.4 compares beat-to-beat values for the simulation against norms listed by Milnor in Mountcastle's *Physiology* [39], [40]. Although any definition of "normal" contains a wide range of acceptable values, those listed are considered typical for a 70 kg adult.

4.3.2 Pulsatile flow waveforms

The pulsatile flow due to the heart beat reveals an exponential time constant for flow between each compartment. This time constant is the same for pressure, volume, and flow curves. The parameter choices were in part determined to provide reasonable time constants (see Section 4.1), which are listed in Table 4.2, page 43.

Comparison of the fine structure of natural pressure, volume, and flow waveforms to those from the simulation reveals similarities and expected differences. Figure 4-4 provides a complete record of pressure, volume, and flow in CVSIM for all six compartments as a function of time. Figures 4-5 and 4-6 show examples of actual catheterizations of normal subjects for

Variable	Reported norm	Simulation result
LV pressure	125/8 mm Hg	113/8 mm Hg
Arterial pressure	120/80 mm Hg	112/75 mm Hg
CVP (& RA pressure)	4 mm Hg	6.8 mm Hg
RV pressure	25/4 mm Hg	23/1 mm Hg
Pulm. arterial pressure	25/10 mm Hg	22/10 mm Hg
Pulm. venous (& LA) pres.	7 mm Hg	12/8 mm Hg
LV stroke work	1430 mW	1050 mW
RV stroke work	290 mW	220 mW
Cardiac Output	5000 ml/min	5200 ml/min
Stroke Volume	77 ml	72 ml

Table 4.4: A comparison of normal cardiovascular simulator hemodynamics to a set of norms from the literature [39], [40]. Entries with two values are peak-systolic/end-diastolic values.

comparison. Although the gross waveforms are similar, there are effects presumably due to peripheral pressure reflections and the aortic and pulmonic valve closure dicrotic notch artifacts in the real catheterization data which are not present in the simulation data.

Pressure–volume tracings in the ventricles reveal a hysteresis loop which graphically demonstrates the work of the ventricle in the cardiac cycle. In addition, the end-systolic pressure-volume points under various loading conditions define the maximum elastance for the ventricle. Figures 4-7 and 4-8 provide a comparison between simulated and real left ventricular pressure–volume loops.

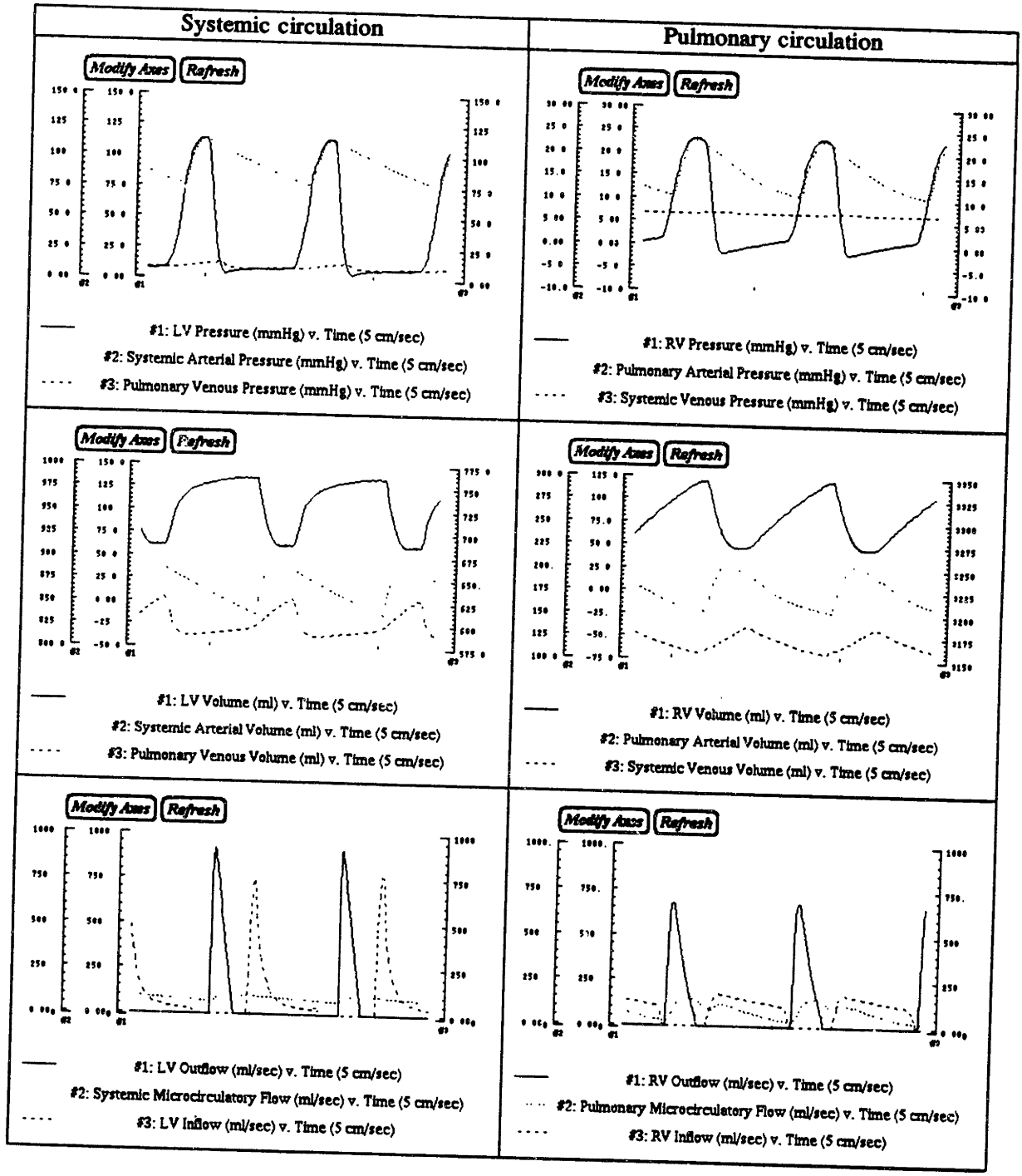


Figure 4-4: Plots from the normal cardiovascular simulation. Top row is pressures, second row is volumes, third row is flow. Left column is systemic variables, right column is pulmonary variables.

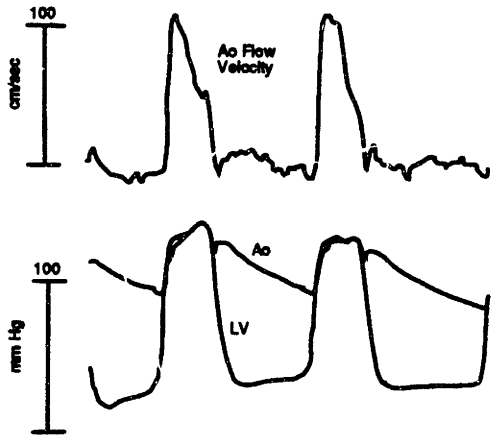


Figure 4-5: Normal patient left ventricular and aortic pressure and flow velocity tracing. Between the first and second systoles the patient begins to contract the abdominal muscles in a Valsalva maneuver, noticeably raising LV diastolic pressure. From Grossman [22]. No time scale was given.

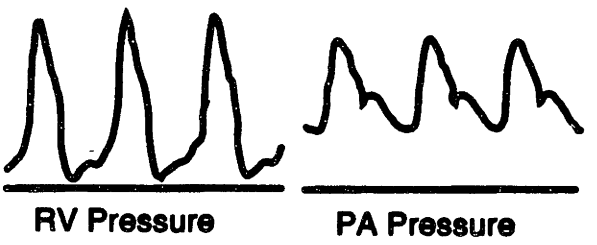


Figure 4-6: Normal patient right ventricular and pulmonary artery pressure tracing, using an intra-cardiac micromanometer. From Laurens [30]. No time or pressure scales were given.

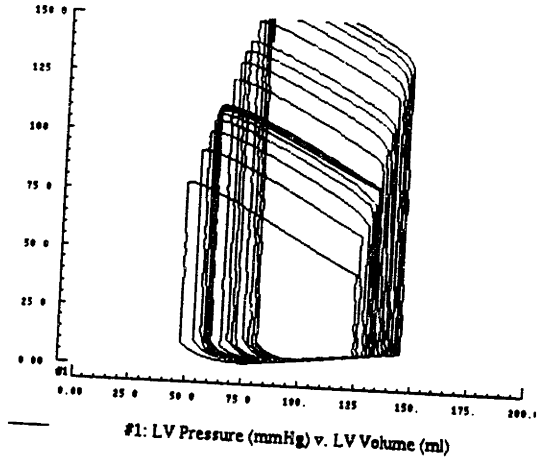


Figure 4-7: Simulated left ventricular pressure–volume loops. Vertical axis is pressure in mm Hg; horizontal is volume in ml. Varied cardiac loading was achieved through changes in peripheral resistance.

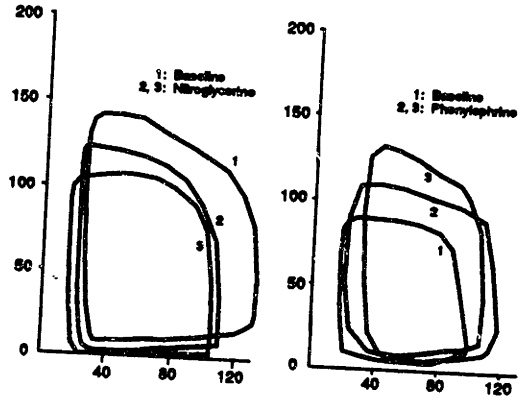


Figure 4-8: Left ventricular pressure–volume loops constructed from gated blood pool scans in two patients. Vertical axis is pressure in mm Hg; horizontal is volume in ml. Vasoactive agents were used to vary cardiac loading. From McKay et al. [37].

Chapter 5

User Interface

The simulation was in need of a user interface which allowed for physiological experimentation on the part of the student. Since this software is utilized in physiology courses in which students are not expected to know techniques of numerical simulation, it was desirable to hide the implementation details, and to allow the students to manipulate the system and make observations based on principles of physiology and on the analytic approach to the model as discussed in class.

The objectives of the user interface design, a description of the implementation effort, and a discussion of implementation issues are provided to aid others who wish to produce graphical user interfaces to other simulations. The development process has become much easier with the maturation of the interface platform and the advancement of workstation technology. Future efforts should be more efficient, due in part of to the production of some of the user interaction tools developed for this simulation.

5.1 User interface design objectives

In order to maximize the utility of the user interface to the simulation, a set of design objectives was developed:

1. The user interface should incorporate the structure of the model in an intuitive manner, such that the correspondences between the model as taught in the classroom and the mechanisms for manipulating the simulation are transparently clear. Interaction with a familiar graphical representation of the model would provide such a correspondence better than a lengthy set

of commands and variable names. This objective was also addressed by previous teaching models [54], [47].

2. Interactions with the program should reflect the dynamic nature of the system. A real-time simulation is thus desirable. This would be especially useful for short-term models such as this, since the time scales for simulated data and observation time are on the same order. It would not generally be useful to model a process in real time if the interesting time frame of the process were less than a second or more than several minutes.

3. The student must be able to monitor any hemodynamic variable of interest, and be able to plot variables together as the system evolves through time. In order to maximize flexibility, the student should be able to choose any set of variables to plot in any combination at any scale. Several different plots should be viewable simultaneously.

4. Parameter values must not be rigidly fixed during a simulation, but should be freely accessible to the student for modification. The student should be able to configure the parameters to mimic any conceivable hemodynamic pathology within the scope of the model. Parameter changes should occur instantaneously, so that transient responses may be observed. There should also be mechanisms for observing steady-state and average behavior.

5. Instructors should be able to set up simulated patients for student assignments, providing a case history and an automatically-loaded set of hemodynamic parameters for each patient. This is intended to encourage students to take a clinical approach to the simulated patient.

6. Students should be able to make printouts of their graphs for presentation to the instructors and other members of the class.

5.2 User interface implementation

These design objectives were addressed by the implementation of a graphical user interface on the X Window System, using the X Toolkit, Athena Widget Set, and custom-programmed widgets. Additional details on the user interface are presented in Appendix A.

5.2.1 X Window System and X toolkit

The X Window System, or simply "X," is a protocol for communication between an application program and a graphical user terminal. The X terminal consists of one (or more) graphics displays, a keyboard, and a pointer device such as a mouse. Application programs occupy independent windows on the display. The X protocol has no built-in interaction policy; it merely provides the programming interface to create and manipulate windows, draw text and graphics, and receive input from the keyboard and pointer (mouse).

In order to provide a user interface, a set of interaction rules must be adopted, assigning meaning to button presses and pointer motion over each area of the display. The top-level application windows are manipulated by a *window manager*, which provides for a cooperative sharing of the terminal resources to each independent application. The applications themselves must define their own interaction rules pertaining only to the areas of the display created by them.

Application programs commonly are built with the aid of user interface toolkits, which manage the low-level formatting and interaction semantics of windows. The standard user interface toolkit for X is the X Toolkit Library, called "Xt." Xt provides a pseudo-object-oriented programming paradigm in the C programming language. Numerous Xt interface objects, called *widgets*, have been developed and collected into widget libraries for the convenience of the application programmer. One such library, the Athena Widgets, was used in the new version of the Cardiovascular Simulator (CVSIM) to define much of the interaction semantics. There were no acceptable predefined widgets to handle dynamic graphs or graphical parameter manipulation, so widgets were designed from scratch for those purposes.

5.2.2 Circuit diagram interaction window

In order to incorporate the structure of the model into the graphical interface, a multiple-level block diagram of the electrical circuit hemodynamic model was implemented. When the user requests to observe a variable or access a parameter, a simplified block-diagram of the cardiovascular system is displayed (Figure 5-1), similar to the schematic used by Sah (Figure 5-1, page 25). This diagram divides the circulation into eight regions: left and right hearts, systemic arteries, microcirculation, systemic veins, pulmonary arteries, pulmonary microcirculation, and pulmonary veins. In addition, an area for general system parameters, such as hr and P_{th}, is provided.

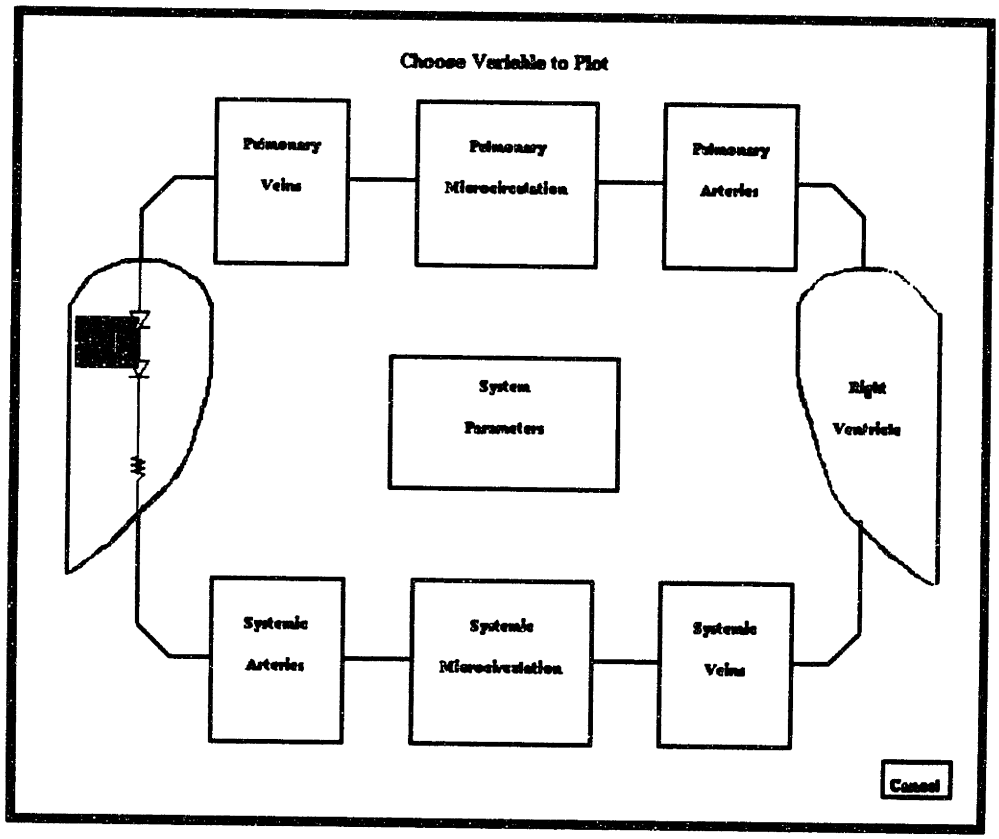


Figure 5-1: Circuit diagram interaction window. The user has clicked the pointer over the left ventricle, and has chosen the capacitor element. If the user next clicked the button on this capacitor, a small menu would pop up offering ventricular pressure, volume, and capacitance as possible choices for plotting.

When the pointer is passed over any of these regions, the region is highlighted. Clicking the left pointer button on a highlighted region selects that region and moves the user down to the next level of specification: electrical circuit elements. The region block in the diagram is replaced by the detail of individual circuit components in that region. For instance, the left heart region displays a variable capacitor connecting the LV pressure node and transthoracic pressure, with heart valve diodes adjacent to the LV pressure node. There is also an outflow resistance connecting the outflow diode to the systemic arteries.

Placing the pointer over a circuit element with accessible parameters causes that device to highlight. Clicking on a highlighted circuit device reveals the final level of specification: a menu of parameters pertaining to that circuit component. For example, clicking on the left ventricular capacitor reveals a list of parameters including minimum and maximum LV capacitance (elastance), and LV zero-pressure volume. If the request is to plot a variable, instantaneous left ventricular pressure and volume are in the list.

Thus whenever the user manipulates the model, whether accessing a model parameter or monitoring a simulation variable, he or she is reminded of the structure of the system and of the place of that parameter or variable in the system.

5.2.3 Plot widget

In order to develop a dynamic simulation, it was necessary to develop a continuously-updating plot widget. The plot widget which was implemented follows the Xt widget guidelines to maintain compatibility with other Xt widgets. It displays up to three variables on independent vertical axes plotted through time or another variable on the horizontal axis (Figure 5-2). When the horizontal axis is time, the plot can update in one of two modes: "strip chart" or "scope." The "strip chart" mode creates an illusion of a strip of paper rolling out of a recording machine and across the display from left to right, disappearing at the left edge of the window. The "scope" oscilloscope mode draws over old data from left to right, starting at the left again when the right margin of the window is reached. The non-time axes have an optional "auto-scale" feature which will redefine the boundaries of the plot whenever a waveform peak is clipped by the edge of the graphing area. Plots of one variable against another, for instance ventricular pressure – volume ($P - V$) loops, can accumulate old tracings and obscure new data. To erase this old data, a "Refresh" button is

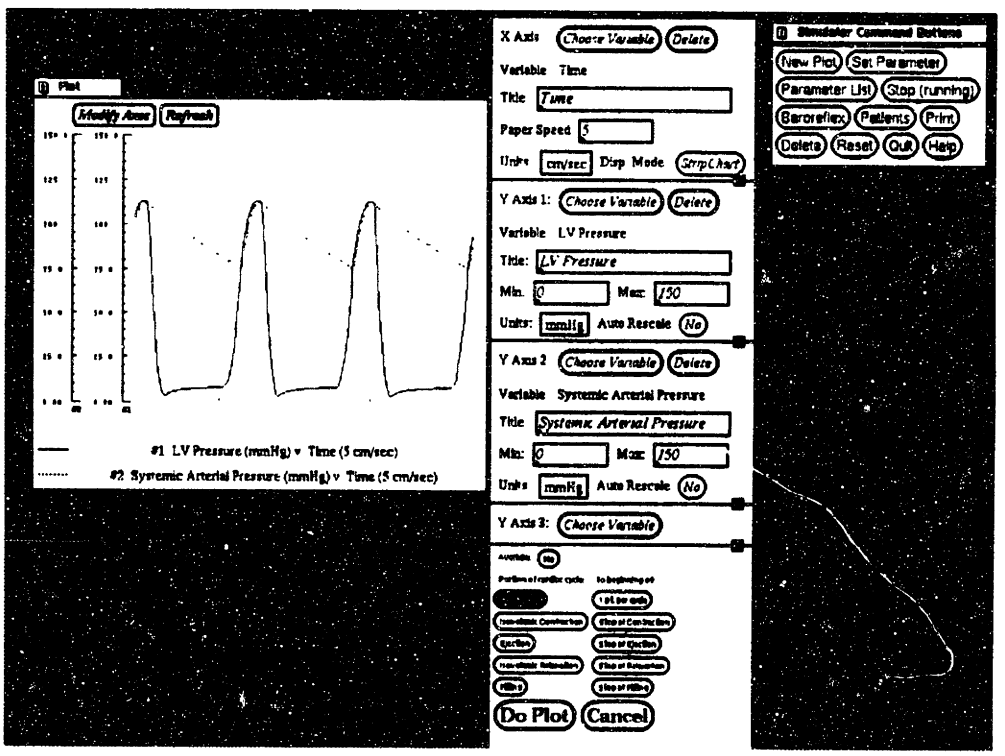


Figure 5-2: The "New Plot" and "Modify Axes" buttons bring up a menu for selection of plot parameters. After variables are chosen from the circuit diagram for each desired axis, the axis parameters such as scaling, paper speed, and labelling may be modified from their parameter defaults. The plot on the left, of left ventricular pressure and systemic arterial pressure versus time, is being modified through the use of the menu in the middle.

provided. Plots can be continuous, or can graph one point per cardiac cycle, or a specific subset of each cardiac cycle, for instance only the ejection phase.

Once created, the plot widget reacts to several sources of input:

- New data is given to the plot widget by an external function. The plot widget updates the screen as necessary to reflect the new data points.
- Events from the display, such as a change in the size or visibility of the window by the window manager, cause the plot to update itself accordingly.
- Events from the pointer cause the plot to react to user requests:
  - Motion of the pointer in the graphing area causes a movable crosshair with coordinate values to appear, for instrumented readings of tracing values. Pressing the left pointer button causes the crosshair and values to be copied onto the plot graph at that position.
  - Pressing the left pointer button on the "Refresh" or "Modify Axes" buttons causes the graph to be erased or the plot-creation menu to appear, respectively.

This implementation of the plot widget provides the ability to program an event-driven user interface which is updated incrementally by simulation data while continuing to be reactive to user input.

5.2.4 Parameter modification widget

A parameter modification widget was developed to allow the user to graphically manipulate parameter values with the pointer. A number line is presented, bounded by minimum and maximum allowed values for a parameter, and a thick arrow pointing directly at the current value, which is shown above the number line (Figure 5-3). Moving the pointer over the number line produces a thin caret cursor which scans over alternative values on the number line. When the pointer button is pressed, the parameter widget issues notification to the main program and moves the thick arrow to the new location. A new parameter value may also be typed in from the keyboard. In this case, pressing the return key completes the entry and moves the thick arrow.

Pointer positioning has a pixel-sized granularity which can lead to unfortunate choices for number-line-chosen parameter values; for example, values of 0.997 and 1.058 might be available,

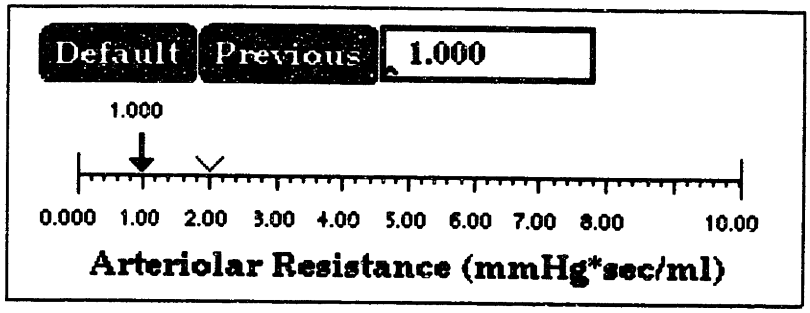


Figure 5-3: The parameter choice number line interaction widget. Instantaneous parameter changes are made by clicking the pointer over the desired new value.

but 1.000 unattainable because there is no pixel to choose between 0.997 and 1.058. Because the number line is shown with round values at tic marks, it is desirable to have those tic mark values selectable. The actual pixel dimensions on the number line are used to establish a working selection granularity, and the available values are chosen so that they increment by 1, 2, or 5 in the last decimal place, depending on the scale, without skipping as the pointer is dragged down the number line.

Because notification is made to the main program at the precise instant of changing the thick pointer position, the simulation can be updated and transient behavior noticed on any running plots while the parameter is still available for further modification.

5.2.5 Plot setup menus

A menu of axis choices provides a graphical interface to create new plots and to modify existing plots. Each of the three possible vertical axes and the horizontal axis are defined by a variable choice, minimum and maximum points or paper speed, and a title notation for the plot. Choosing a variable from the circuit diagram fills in default values for all fields, which are then modifiable by the user.

5.2.6 Patient case environment

A patient case database was added to the user interface which allows the definition of parameter sets accompanied by a textual case history. The parameter modification number lines are altered

to reflect *percentages* of the patient's original parameter values rather than the real parameter values. This keeps the parameter values a mystery[1], so that students are forced to use clinical reasoning to diagnose the disease based on the case history and hemodynamic simulation alone.

An instructor's manual was written to aid the instructor with the development of new patient cases, and to provide documentation for program maintenance.

5.2.7 Printing results

The X utility "xdpr" is used by the program to print copies of plots made on the screen. A specific plot of interest or the entire X display screen can be printed on a laser printer at the touch of a button.

5.3 User interface implementation issues

In the course of the implementation effort, several issues surfaced which deserve attention. These included problems with the numerical solution algorithms, simulation efficiency and response time, and problems specific to implementation details on the chosen platform, M. I. T. Project Athena.

5.3.1 Numerical simulation difficulties

There were technical difficulties with the numerical integration method for the mathematical model. The simulation can fail to converge if the time slice is too long during early systole and early diastole, or if the ventricular capacitance changes too rapidly, or if the ventricular outflow resistance is too low, or if any of a number of other factors causes a rapid change in ventricular pressure. This is due to the stiffness of the system: some of the dominant time constants in early and late systole are very short. The simple Runge-Kutta procedure is known to behave poorly when used on stiff systems, because time steps appropriate for the rest of the system are too large for a subset of the simulation time.

[1] Students can usually *calculate* unknown patient parameters by means of hemodynamic measurements in order to quantify the disease process being studied.

I made attempts to improve simulation efficiency by

- reducing the stiffness of the system, and
- tailoring the integration method.

To reduce the stiffness of the system, the difference between the shortest and longest time constant must be reduced. The shortest and most destabilizing time constants are the RC time constants for ventricular ejection, which are around 2 ms for both ventricles. Fortunately, the pressure gradient between ventricle and artery is normally very small, so a simulation time step of 2 ms is not quite necessary. However, a rapidly changing ventricular contractility can increase this gradient, leading to convergence failure if the step size is too large. Our $C_l(t)$ and $C_r(t)$, designed to mimic physiological experimental results (see Section 4.1.5, page 38), were initially more steep in early systole than data on human $((1/P)dP/dt)_{max}$ indicate. Reducing the maximum slope of the capacitance and removing discontinuities in $dC_l(t)/dt$ and $dC_r(t)/dt$ increased the experimentally-determined minimum time step dramatically; however, the slope of the ventricular pressure tracing was extremely sensitive to such changes. Therefore, a maximum ventricular slope was selected (see Section 4.1.5, page 38) which was within reasonable limits for $((1/P)dP/dt)_{max}$ and which produced an acceptable appearance for ventricular pressures.

Increasing the outflow resistance or increasing the instantaneous ventricular capacitance would directly increase the RC time constant, which would also increase the minimum time step. Unfortunately, increasing the resistance by a significant amount would cause nonphysiologic outflow gradients to appear, indicating an arterial stenosis. An increase in capacitance would decrease contractility, and hence pressures, below the desired level. Therefore, these parameters have not been modified.

To tailor the integration to the model, I experimented with varying time-step methods. It seemed sensible to make the time step dependent on the local slope of the ventricular capacitance curve, but I was never satisfied with an ad-hoc method that was both robust and efficient in for all simulated patients. However, a general adaptive-timestep technique for stiff systems [48] was recently found to be robust in the face of massive parameter deviations, even with simulation parameters producing time constants five orders of magnitude shorter than normal. This adaptive technique compares two fourth-order Runge-Kutta steps of different sizes to determine the max-

imum error in the state variables. This error is used to control the step size. In this unclocked pseudo-real-time simulation, adaptive timestep integration causes a fast spurt of plotted data in diastole and slower plotting in systole, which might be misleading to students; therefore, the original simple Runge-Kutta simulation is left as a compile-time option. Another option is to clock the plotting of data to the screen from a buffer; however, a faster computing platform would be needed to fill such a buffer as fast as it is emptied, and the illusion of real-time performance would thus suffer with the current hardware.

5.3.2 Interactive responsiveness of program

Data from the program is currently produced at the maximum rate and displayed as soon as it is computed. For IBM RT and DEC VaxStation 3100– class workstations, running at 3–5 million instructions per second, this produces output at a maximum rate of approximately 0.75 s of data per second, or three-quarters real time. Running on a faster computer, such as a Sun Sparc Station, produces data several times faster, making real time operation feasible. If such fast workstations were to be used as the delivery platform for student use, the data could be clocked to real time, which would convert the program from a pseudo-real-time simulation to a real-time simulation. A side benefit of such a clocked-output system would be improvement in the responsiveness to user input. Currently, a running simulation with many active plots causes a backup of unplotted data in the X server queue. As a result, the reactions on the screen of pointer motion, button clicking, and keyboard entry can be delayed significantly. For instance, when one moves the pointer over a push button on the screen, the button should automatically highlight to indicate its selection. If the pointer is dragged over several buttons, as is usual since they are clustered together, the user may see, for instance, several seconds of continued simulation followed by the repeated highlighting and unhighlighting of several buttons. If a button has been pressed in the meantime and no response was noticed, the user may then see his button presses received and new windows appearing, but all separated from the requesting action by enough time that the user may not realize what caused the new windows to appear.

Activities could be prioritized such that user interaction has precedence over plotting of simulation data. Though this has been done in the top-level program, the layering of event queues and the nature of the X protocol make useful prioritization impractical: all pending X

events are purposely dispatched by the program before any additional simulation data is produced, so that user input will be processed in a timely fashion. Further control would require knowledge of the server's actions. The only way to verify that an action is taken on the server before enqueueing an additional instruction is to issue an `XSync()` command and wait for a reply. However, blocking for `XSync()` regularly would slow execution dramatically and still would not guarantee that a user input action is serviced immediately before any more simulation activity is plotted.

As long as the number of active plots is kept to a reasonable number, the response is quick, so that reactions to pointer motion and button presses remains intuitive and understandable. Thus the objective to provide a real-time simulation with a dynamic user interface has been achieved.

5.3.3 Difficulties with the computing platform

Aside from difficulties associated with the speed of the processor, there were several recurrent problems that hampered the software development process as it progressed slowly from 1986 to 1990. These problems were associated with various aspects of the nascent Project Athena computing environment:

- Frequent incompatible changes to the X Window System, Xt Toolkit, and Xaw Athena Widget Set.
- Bugs in the X Toolkit and Athena widgets.
- Frequent changes in the Project Athena operating system utilities and network resources.
- Problems making bitmapped printouts to an "unsupported" printer, the IBM 3812 Page-Printer in the E25-131 cluster.
- Problems with network access from E25 before and after the cluster's connection to the rest of Project Athena.
- Problems porting the software to the IBM RT High C compiler, and tracking that compiler's development to the current version 2.1y.
- A bug in the dbx debugger on the IBM RT which causes unreliable reporting of the values of double-precision floating point variables from within the debugger.

These problems were largely resolved after the first few releases of the X Window System Version 11, but significantly hampered development. The conversion of the program X Version 10 to X Version 11 was especially time consuming, as were compiler bugs in the early versions of the High C compiler for the IBM RT/PC. All of the problems listed above have been competently managed by the Project Athena staff and the hardware manufacturers, such that they pose no barrier presently to further development.

Chapter 6

The Baroreflex Control Model

The CVSIM program responds appropriately to dynamic parameter modifications of capacitances, resistances, heart rate, blood volume and trans-thoracic pressure. However, the system as described thus far operates without feedback control.

In order to produce a realistic simulation of a disease process, the student must take care to alter all parameters known to be under the influence of physiological feedback mechanisms. For instance, a sudden loss of blood would cause a reactive tachycardia in a normal individual, but in the uncontrolled simulation, heart rate remains constant until modified by the user. In this mode, the student must become the cardiovascular control system.

This chapter describes an addition to the hemodynamic model in the form of an automatic feedback control system: the *arterial baroreceptor reflex*, or *baroreflex*. The baroreflex simulation was designed as a simple, intuitive extension of the basic hemodynamic system as implemented in CVSIM.

6.1 Homeostatic mechanisms in the cardiovascular system

The mechanical properties of the cardiovascular system are modulated by a network of homeostatis-maintaining feedback mechanisms. They include both hormonal and neural effects on the heart, peripheral microvasculature, veins, and blood volume in feedback response to pressure, volume, and chemical proprioceptors within the cardiovascular system, the pulmonary system, and the brain. Only the short term control of arterial pressure, acting over seconds to minutes, will be

considered here; mechanisms with a slower time course, including hormonal and renal compensatory mechanisms and local tissue oxygenation blood flow reflexes, are ignored.

The arterial baroreceptor is not the only component of short-term control of the circulation. There exist poorly understood so-called "low-pressure" baroreceptors in the pulmonary vasculature and the atria which are known to regulate blood fluid volume and are thought to influence heart rate[1] and possibly peripheral venous tone [36], [33], [26]. Because of difficulties in determining parameters for such a poorly-understood system, and because the systemic arterial control model seems to be more important for short term control of the systemic circulation, we will ignore these low-pressure reflexes in this study.

6.2 The baroreflex

The arterial baroreflex is the principle mediator of short-term control. Carotid artery and aortic stretch receptors transmit to the autonomic nervous system a signal (S) which is modulated in rough proportion to arterial blood pressure fluctuations (Figure 6-1). A fast parasympathetic reflex arc through the vagus nerve tightly controls heart rate through the sino-atrial (SA) node, while slower sympathetic fibers modulate the strength of ventricular contraction and peripheral vascular tone in addition to a slow contribution to heart rate.

Many approaches have been taken to characterize components of this system. Transfer functions for the baroreceptor, SA node, and other portions of the system have been approximated in open-loop and closed-loop conditions in animals and in man [52], [51]. In the early 1960's Noordergraaf, Beneken, and others [44], [3], developed complex analog computer models incorporating the baroreflex. More recently, DeBoer, Karemaker, and Strackee [14] derived a simple proportional control model based on physiological concepts which appeared suitable for addition to our simulation. A more general mathematical approach is taken by Appel, using parameter estimation techniques to derive transfer functions between heart rate and blood pressure [1]. Parameter estimation methods, however, do not identify the physiologic origins of the control equations. I therefore developed a continuous-time model which uses an approach similar to the beat-to-beat model of DeBoer et al.Although DeBoer et al. do not use the terminology of digital

[1] The so-called *Bainbridge reflex* mediates heart rate increases in response to increased atrial filling.

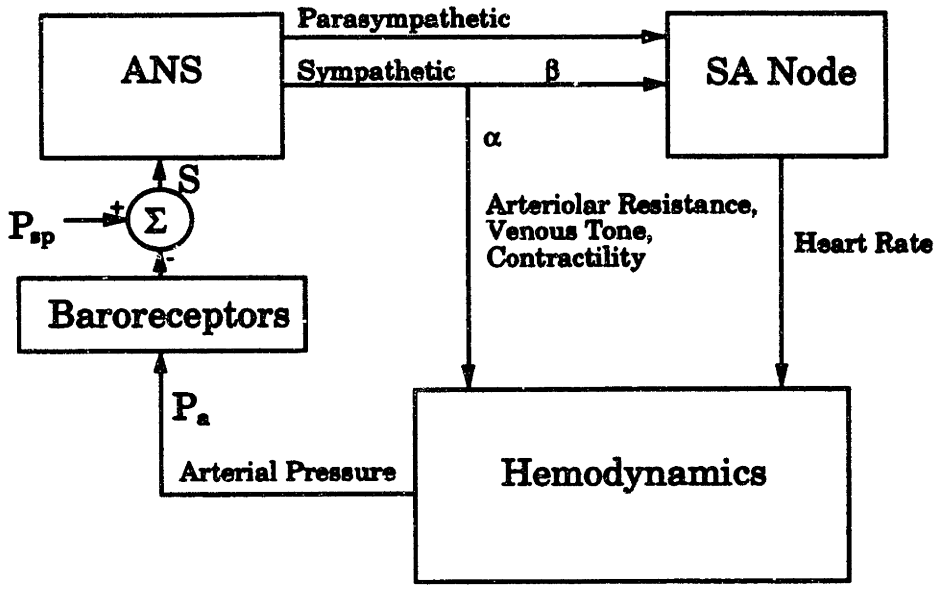


Figure 6-1: A general outline of the baroreflex control loop. The autonomic nervous system (ANS) receives pressure deviation signals S from arterial pressure receptors and exerts homeostatic controlling actions on the hemodynamic system. (Adapted from Madwed [34].)

signal processing, their work will be restated in those terms where appropriate for simplicity of comparison to my model.

6.3 The DeBoer model

The DeBoer et al. cardiovascular system model is based on a set of beat-to-beat difference equations for the control system and a *windkessel* model of the circulation. Eq. (6.1) describes an "effective blood pressure deviation" measure, s', which accounts for the saturation of baroreceptor response at extreme pressure deviations:

$$s'[n] = 18 \tan^{-1} \frac{s[n] - s_0}{18} \tag{6.1}$$

where the blood pressure setpoint, s_0, is set to a target pressure for $s[n]$, the systolic pressure. This arctangent mapping preserves the baroreceptor sensitivity around the setpoint, while limiting the maximum signal to $\pm 18\pi/2 \approx \pm 28$ mm Hg deviation from the setpoint.

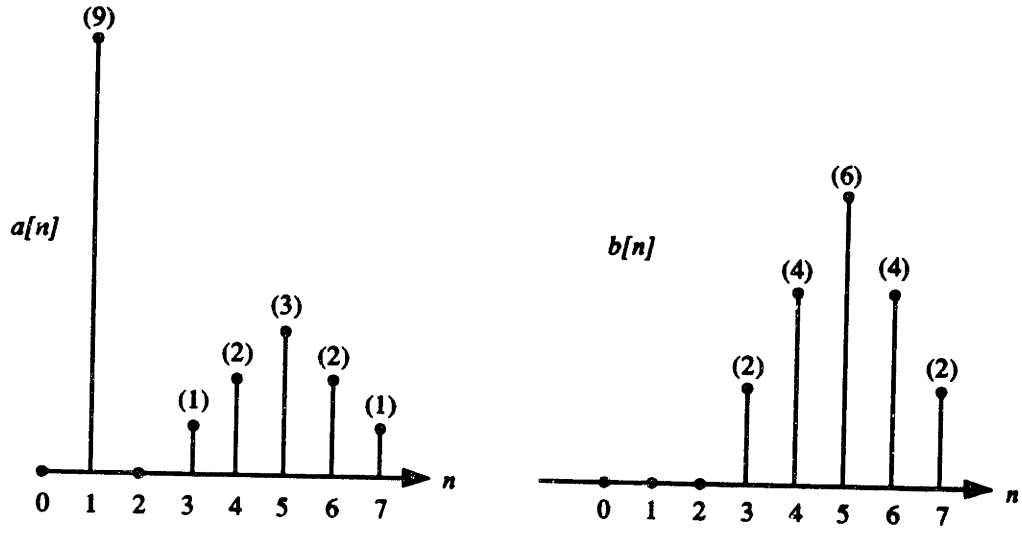


Figure 6-2: Unit sample responses for heart rate ($a[n]$) and *windkessel RC* time constant ($b[n]$) feedback, from the DeBoer et al. model [14].

The effect of the baroreflex on heart rate is described as a unit sample response for the one-beat-per-sample discrete model. These unit sample responses, $a[n]$ and $b[n]$ in Figure 6-2, are convolved with the $s'[n]$ signal to generate a reflex signal for altering the heart rate and the *windkessel* time constant. Eq. (6.2) shows the calculation of heart rate based on the pressure during previous beats. The inter-beat interval $I[n]$ is adjusted by the current $s'[n]$ signal filtered through convolution with $a[n]$:

$$I[n] = I_0 + \sum_{k=0}^{7} a[k]\, s'[n-k] \tag{6.2}$$

where the $a[i]$ in Figure 6-2 describe a fast vagal response ($a[1]$) equally weighted with a slower sympathetic response ($a[3]$ through $a[7]$). Similarly, Eq. (6.3) describes the effect of baroreflex on the arterial outflow time constant:

$$T^{*\prime}[n] = T^* - \sum_{k=0}^{7} b[k]\, s'[n-k] \tag{6.3}$$

where T^* is a baseline RC constant for the peripheral circulation and $T^{*\prime}[n]$ is the calculated RC time constant for beat n.

The DeBoer et al. model included approximation equations incorporating venous return, cardiac contractility, and peripheral circulation which are not applicable to this simulation because these parameters are already modelled individually in CVSIM, whereas they were treated in combined forms by DeBoer. The first of these equations is the following:

$$P[n] = \gamma\, I[n-1] + c_1$$

This computes the next beat's pulse pressure $P[n]$ as a function of heart rate and thus venous filling time, according to Starling's law. Pulse pressure in the CVSIM model is determined instead through the time-varying elastance simulation which causes a pulse pressure dependence on cardiac filling, contractility, and the state of the systemic and pulmonary vascular beds. DeBoer et al. also used a *windkessel* model of the peripheral circulation:

$$D[n] = S[n-1]\, e^{-\frac{I[n-1]}{T[n-1]}}$$

in which the decay of arterial pressure to an end-diastolic value $D[n]$ from the previous systolic pressure $S[n-1]$, over the diastolic period $I[n-1]$, is dependent only on $T[n-1]$, the peripheral RC time constant. Again, the CVSIM model already directly accounts for this behavior in a more detailed hemodynamics simulation.

6.4 Control system design

Four limbs of the baroreflex were modelled for implementation in the CVSIM program: heart rate, cardiac contractility, peripheral resistance, and venous tone (see Figure 6-1). The controller is modelled as a set of parallel linear filters. Parameters for this model were determined from published findings on open-loop gain and receptor, nerve, and end-organ delay timings in humans. The control model was verified by comparing closed-loop gain, time response, and experimental stability to human laboratory data.

6.4.1 The baroreflex model

The CVSIM program models the pulsatile nature of the cardiovascular system with time steps on the order of 10 ms, whereas DeBoer et al. used a beat-to-beat model.[2] Eqs. (6.1), (6.2) and (6.3) were incorporated directly into our model, with $s[n]$, systolic pressure indexed by beat n, replaced by $s(t)$, current intra-beat pressure indexed by time t at a small time step granularity, which is a more realistic approximation of input to the baroreceptor. The $a[n]$ and $b[n]$ from Deboer et al. were replaced by more accurate impulse responses for sympathetic and parasympathetic responses (see Section 6.4.2, page 75). A 30 second history of $s'(t)$ values is kept for the convolutions, which are performed for periods of 0.5 s, rather than at every CVSIM timestep. The arterial pressure signal is averaged to provide values for successive 0.5 s periods as input to the control filters. This coarser timestep for the control system (see Section 6.5.3, page 80) was used to increase simulation speed while maintaining an accurate feedback response.

Two implemented reflex limbs were not directly addressed by the DeBoer et al. model: the well-known control of (1) ventricular contractility and (2) venous tone [24]. Since $C_{l,sys}$ and $C_{r,sys}$ correspond to the capacitance[3] of the ventricles during systole, a proportional modulation of these values was developed and scaled to obtain realistic minimum and maximum contractility:

$$C_{l,rsys}(t) = C_{l,rsys0} + \int_{k=0}^{30} c(k)\, s'(t-k)dk \tag{6.4}$$

This formula mimics Eqs. (6.2) and (6.3) for continuous time, but applies control to contractility with an impulse response $c(t)$ to mimic the β-sympathetic response time.

A similar control model is proposed for peripheral venous tone. The greater portion of volume in the peripheral veins is modelled by the zero-pressure filling volume, or charge offset of the venous capacitor (V_{v0}). The linear capacitance contributes relatively less to venous blood storage capacity: 2000 ml of the average 3200 ml in the systemic venous compartment is in the unstressed (zero-pressure) filling volume (see Section 4.1.6, page 41). Therefore, a linear modulation of V_{v0} would regulate venous blood pooling appropriately to mimic α-sympathetic

[2] DeBoer et al. [14] used one simulation step per beat.

[3] Recall that capacitance is the reciprocal of elastance, a measure of contractility.

control of venous tone:

$$V_{v0}(t) = V_0 + \int_{k=0}^{30} d(k)\, s'(t-k)dk \tag{6.5}$$

Thus the baroreflex is modelled as a bank of four parallel linear filters, described by unit sample responses $a(t)$, $-b(t)$, $c(t)$, and $d(t)$, with the "effective arterial pressure deviation" $s'(t)$ as input. These filters provide feedback control to hemodynamic system parameters hr, $C_{l,sys}$, $C_{r,sys}$, R_a, and V_{v0} to complete the baroreflex model. All that remains is to design the individual filters, which are composed of temporal responses scaled to overall open-loop gains.

6.4.2 Filter temporal response design

Delays in the baroreflex response

Because of the beat-to-beat nature of DeBoer's system, the delays inherent to neural feedback were modelled inaccurately. Returning to the basic physiology, let us investigate the sources of delay in the baroreflex:

Blood transport delays Delays associated with the traversal of a bolus of blood through the system are incorporated to a first approximation by the uncontrolled lumped parameter model. Transport delays are based on filling times for each compartment of the system (see Table 4.2.). Pressure wave propagation through elastic tubes is not modelled; however, this characteristic time is less than 100 ms and is thus ignored.

Baroreceptor and efferent nerve time responses These delays are short (100 to 300 ms), and are common to all limbs of the baroreflex.

Central nervous system processing The most active CNS portion of the reflex arc is short (less than 10 ms) and therefore ignored.

Efferent transmission and effector organ response The bulk of the delay occurs here, and varies markedly with limb of the autonomic nervous system and effector organ.

Timing data for P_a to effector organ activity was obtained from Borst and Karemaker [9], Madwed [34], and Berger et al. [5]. Borst and Karemaker [9] measured the latency for initial response to electrical stimulation of the carotid sinus nerves, finding a 0.55 second delay for

Nerve Type	Latency (s)	Peak action (s)	Final response (s)
α-sympathetic	2.0	5.0	30.0
β-sympathetic	2.0	5.0	30.0
Parasympathetic	0.5	0.5	1.0

Table 6.1: Delay for baroreflex response to a step change in P_a. The parasympathetic timing shown includes a delay representing the exitation–contraction coupling time for heart rate changes.

changes in heart rate, and a 2–3 second delay for diastolic pressure change, which was thought to reflect a 2–3 second delay in peripheral resistance reduction. These delays correspond to parasympathetic and sympathetic response times, respectively. Madwed [34] reported a 3–5 s latency for β-sympathetic response, and and a 5–8 s latency for α-sympathetic response. Time until 100% response was 10–15 s for β-sympathetic and 20–30 s for α-sympathetic signals. Berger et al. [5], from the same laboratory, indicate that the difference between Madwed's α-sympathetic and β-sympathetic timings, which were inferred from the time difference between cardiac inotropic response and peripheral resistance response, were likely corrupted by vagal parasympathetic activity in the cardiac experiments, and that α- and β-sympathetic response times are probably the same. Their data forms the basis of the filter design, as will be shown in the next section. Delay times pertinent to the simulation are listed in Table 6.1.

Our model assumes that the onset and offset times of these reflexes are equal; however, it is known that they are not [52]. The onset of a positive sympathetic response occurs over a 15 to 30 s interval, but withdrawal of activity is known to take considerably longer than that, due to slow reuptake of acetylcholine at presynaptic nerve cells. Most other models also make this simplification [4], [43]. The model will respond faster to events which raise arterial pressure than is physiologically accurate, if it is to respond accurately to lowering of arterial pressure. The body maximizes its ability to react quickly to a drop in pressure, teleologically because a rise in pressure is a less immediate threat to life than is a sudden drop in pressure, which could cause syncope.

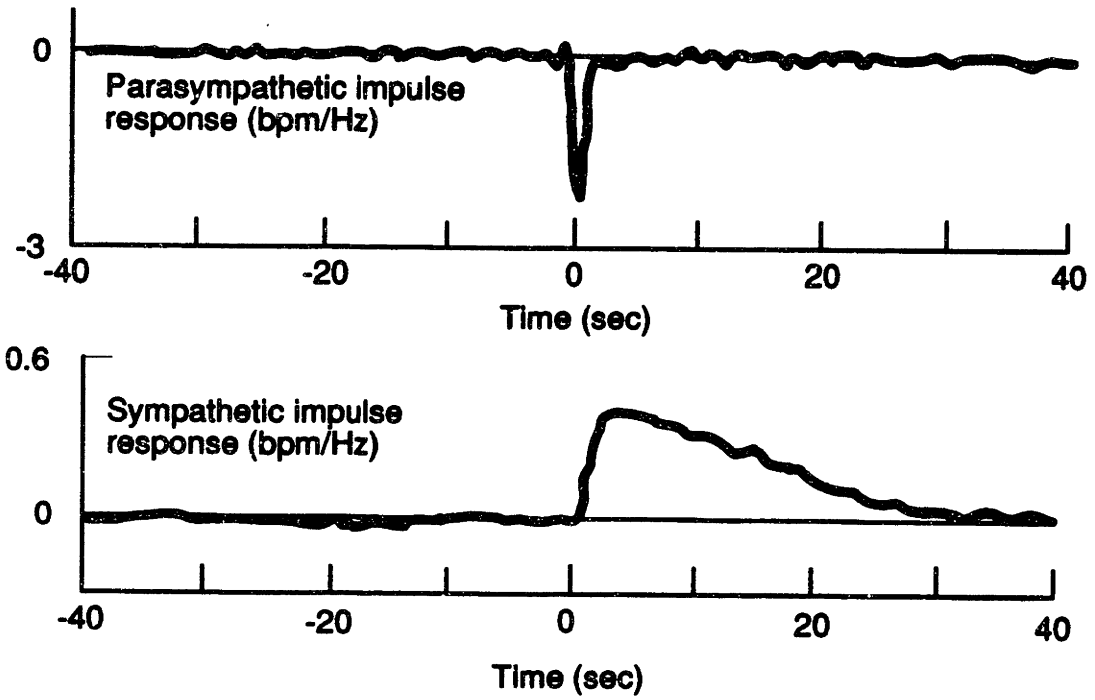


Figure 6-3: Impulse responses to sympathetic and parasympathetic nerve stimulation. From Berger, Saul and Cohen [5].

Impulse responses

The timings in Table 6.1 indicate initial delay, time to maximum slope, and time to final value for a response to a step change in arterial pressure. A more direct implementation results from examination of the impulse response, which is simply the time derivative of the step response. Impulse responses of the heart rate response to sympathetic and parasympathetic stimulation in man were recently derived by Berger et al. [5], (Figure 6-3). These impulse responses were used to design finite impulse response filters to mimic the response of the baroreceptor, autonomic nervous system, and end-organs in effecting the baroreflex. The impulse invariant design technique is ideal for such an application: Figure 6-4 shows a schematic of the designed parasympathetic and sympathetic filters. In actuality, these continuous-time representations were sampled with a 0.5 second period and scaled to sum to the appropriate gain to obtain the required discrete-time unit sample responses.

In comparison to the baroreflex filters, the efferent nerve stimulation experiments of Berger et

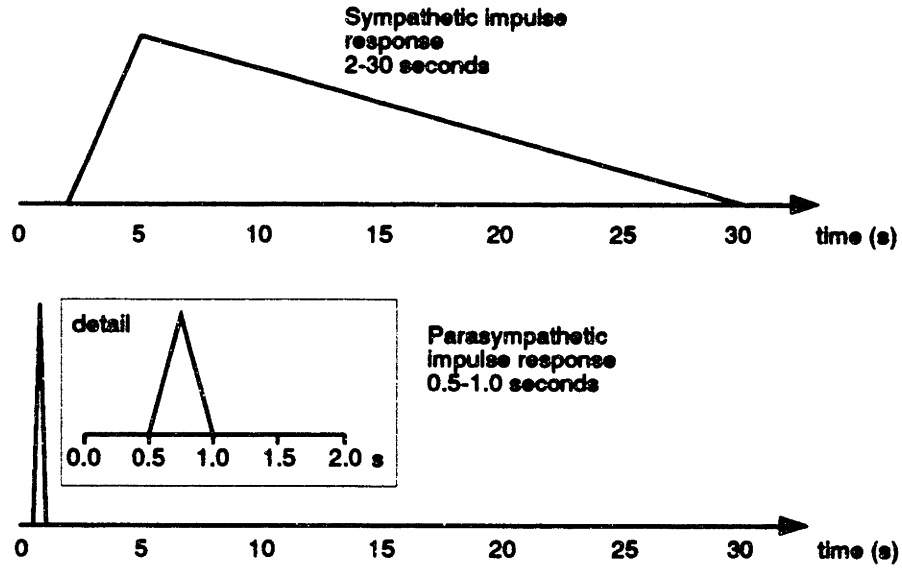


Figure 6-4: Sympathetic and parasympathetic feedback impulse responses for the baroreflex model. A linear combination of these two filters derived from gain studies (see Section 6.4.3) forms the filters for each of the four reflex limbs.

al. do not include the baroreceptor organ response or central nervous system delay; however, as we have seen, the delays associated with these are short (300–500 ms) and relatively fixed, and therefore were modelled as a short pure delay in series with the sympathetic and parasympathetic end-response filters.

The area under each impulse response is scaled according to the calculated gain from Section 6.4.3 to complete the filter design for each reflex limb.

6.4.3 Feedback gain determination

Each limb of the reflex is described by its filter's time response which is scaled to the overall filter gain, or reflex sensitivity. The adopted gain values are listed in Table 6.2.

The baroreflex heart rate sensitivity is known to be in the range of 10 to 20 ms/mmHg [14]. The values from DeBoer et al. of 9 ms/mmHg of β-sympathetic feedback plus 9 ms/mmHg of parasympathetic feedback, for an overall final sensitivity of 18 ms/mmHg, were used in the model.

The baroreflex contractility sensitivity has not been completely worked out. Sagawa and

Reflex Limb	Gain		Nerve Timing
heart rate, hr	18.0	ms/mm Hg	β- & parasympathetic
LV contractility, $C_{l,sys}$	0.007	ml/mm Hg/mm Hg	β-sympathetic
RV contractility, $C_{r,sys}$	0.021	ml/mm Hg/mm Hg	β-sympathetic
peripheral resistance, R_a	0.011	PRU/mm Hg	α-sympathetic
venous zero-pressure volume, V_{v0}	26.5	ml/mm Hg	α-sympathetic

Table 6.2: The reflex gain values used in CVSIM.

co-workers [53] reported that the end-systolic elastance $E_{max} = (P(t)/V(t))_{max}$ changes in response to baroreflex activity. This index increased 65% with left stellate stimulation, and decreased 113% through maximal carotid sinus and vagal stimulation; a maximum value of 227% increase was measured during cerebral hypoxia. Based on this data, a sensitivity which produces a maximum of 100% increase in contractility for the maximal limit of baroreceptor withdrawal was adopted.

The baroreflex peripheral resistance sensitivity is not readily available from the literature. However, the *windkessel* model combined with DeBoer's rough sensitivity to change the RC time constant provides a reasonable approach to resistance sensitivity. I have thus defined the resistance deviation as $1/C_a$ times DeBoer's T^* as described in Eq. (6.3).

The baroreflex venous tone sensitivity has likewise little directly applicable data. Guyton [24] has asserted that peripheral venous volume can change as much as 1000 ml due to postural changes alone. To make a first approximation, I have chosen 750 ml as a reasonably physiologic limit on venous volume excursion due to the baroreflex. When the arctangent limiter equation for $s'(t)$ is considered, this translates to a gain of 26.5 ml/mmHg around the setpoint.

The default setpoint pressure s_0 from Eq. (6.1) is set to 98 mm Hg in order to maintain the mean arterial pressure at 99.5 mm Hg for both the normal uncontrolled CVSIM model and the unperturbed controlled model.

6.4.4 Theoretical stability

It is known that the physiological baroreflex is at best only marginally stable: large step inputs in normal subjects reveal an underdamped oscillatory response [52], which is more pronounced with hemorrhage [34]. This marginal stability can also be found with the modelled baroreflex.

The controlled hemodynamic system can become unstable if positive feedback can be achieved through a combination of long error signal delay and large loop gain. The root locus of the complete system could be calculated, since the controlled and controlling system both have a finite number of poles. For this simulation, the controller has 60 poles, corresponding to 60 samples of impulse response, and the controlled system has six, corresponding to the six compartments. Two of those roots trace out a path in time as ventricular elastance changes. It is easily seen that both the controlled system, an RC circuit with a simple topology, and the feedback filters, finite-impulse-response systems, are both intrinsically stable. However, if viewed together as a linear system, discounting the arctangent limiter of the baroreflex, the controlling system can move some of the poles of the controlled system across the zero point on the real axis and into the unstable region if the gain and delay in the feedback loop are high enough.

The incorporation of the soft limiter ($s' = 18 \tan^{-1} s/18$) on the error signal s guarantees that the state variables will not increase without bound; however, this nonlinearity can give rise to chaotic behavior if the system is driven to the nonlinear limits of s'. A complete description of the stability of the system requires utilization of nonlinear control theory which is beyond the scope of this thesis.

6.5 Baroreflex Model Implementation

The baroreflex model was implemented directly on top of the previously-developed CVSIM program. User interface windows were designed to permit dynamic activation and modification of the baroreflex system. All interaction between the reflex model and the hemodynamic plant is accomplished through modification of controlled variables and a heart rate integrator routine.

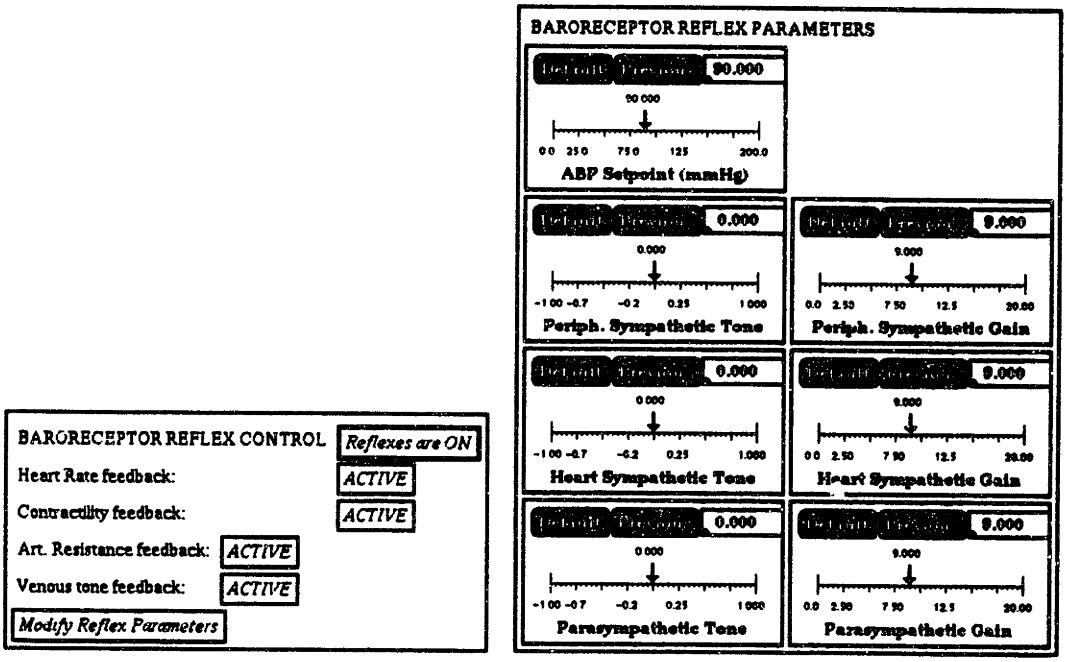


Figure 6-5: User interface windows for the baroreflex control system. The window on the left allows partial or complete activation of the control system. Clicking on the bottom "Modify Reflex Parameters" button produces the window of number lines on the right. The scales used for passive tone are scaled such that ± 1 is the maximum active response from the baroreflex; those for gain are given in units of ms/ml as in the heart rate reflex. The separate gain for each limb of the reflex is currently not available for modification.

6.5.1 User interface

The student can selectively engage or disengage any of the four reflex loops via buttons in a baroreflex control box. A more detailed control is given by a box of number line parameter modification widgets (see Section 5.2.4, page 59) for student modification of arterial pressure setpoint (s_0), heart sympathetic gain, vascular sympathetic gain, parasympathetic gain, and optional tone offsets, to facilitate experimentation. (Figure 6-5). Every time the reflex system is activated, the user is queried whether to use the current history of $s'(t)$ or whether to reset the $s'(t)$ history to the current value. This gives the user some control over the state of the baroreflex at start-up. Carotid sinus pressurization experiments can be mimicked by changing the pressure setpoint in the same direction as the desired baroreceptor pressurization.

When a student selects a hemodynamic parameter to modify by the normal parameter modi-

fication system, such as heart rate, for example, the parameter modified is the heart rate for the uncontrolled system, which is used as the reference heart rate to which deviations are applied. In contrast, the plotted values for heart rate etc. are the controlled parameters, so that the student can directly observe the transient effector actions of the control system.

6.5.2 Hemodynamic parameter control

The hemodynamic parameters $C_{l,sys}$, $C_{r,sys}$, R_a, and V_{v0} are modified by the control system at each control time step (0.5 s), maintaining a consistent simulation by applying the constant volume constraint (see Section 4.2.4, page 47).

In order to add a dynamic heart rate capability to the hemodynamic simulation, the program was modified to accept an external signal to determine the onset of a new beat. The algorithm uses a heart rate weighting function on each Runge-Kutta iteration, summing the values received until the product of the rate sum and the step size exceeds unity.[4] Thus the heart rate signal is integrated over time, producing beats after traversing each unit area under the heart rate curve.

6.5.3 Time step choice

The analog model was quantized and the impulse response converted to a digital filter to incorporate the control model into the hemodynamics simulation. The chosen time step of 0.5 s for the feedback loop simulation may seem large, since the hemodynamics simulation gives waveforms which vary over much smaller time scales. However, the time response of the autonomic nervous system is not nearly as short as the compartmental time constants, and therefore can be simulated in larger steps without losing accuracy. As can be seen from Figure 6-4, the simulated sympathetic feedback occurs over 2 to 30 seconds, and has no fine features of interest below two seconds. The parasympathetic feedback occurs more quickly, over 0.5 seconds. Since it is so fast, the precise timing of the parasympathetic branch is not important to the general behavior of the control system. A simple model of the parasympathetic feedback can be obtained by summing the previous 0.5 seconds of the pressure signal. It is not necessary to sample the impulse responses at the Nyquist frequency, because the fine high frequency structure is not of interest.

[4] This is an IPFM model.

Because calculation of new parameter values and parameter update constraints takes valuable processor time away from the rest of the simulation, a time step of 0.5 seconds has been used for the feedback filter implementation.

6.6 Control System Verification

The control system implementation was tested to verify its closed-loop behavior by comparing closed-loop gain, time responses, and stability against human experimental data where available.

6.6.1 Steady state closed-loop gain

The baroreflex acts to maintain blood pressure homeostasis; this homeostatic tendency can be investigated through studying the closed-loop gain of the system. Since the baroreceptor responds to the pressure difference across the blood vessel lumen, an increase in the neck tissue pressure (NTP) would cause a pressure-loss signal to be transmitted; this would result in a reflex action to raise blood pressure.

Mancia et al. [35] measured the closed-loop gain of a system defined with outputs of mean arterial pressure and heart interval[5] and input of externally-controlled neck tissue pressure. Pressure was applied via a neck chamber to achieve expected NTP in the range of ± 50 mm Hg for periods of 2 minutes at a time. The NTP was calculated as 86% of positive and 64% of negative neck chamber pressures, in accordance with earlier studies on external pressure transfer to the tissues. Onset of pressure changes was rapid: the neck chamber reached 90% of the target pressure within 1 s of application, approximating a step change. Data were collected from experiments on 11 subjects and regression statistics computed.

A similar experiment was conducted with CVSIM for comparison: mean arterial pressure and heart interval were measured after step changes of s_0 in increments of 10 mm Hg to ± 40 mm Hg in the normal simulation, and regression statistics calculated separately for positive and negative changes. A comparison of the average linear regression slope coefficients to a simulation on CVSIM is given in Table 6.3. The Mancia et al. data reveal an asymmetric late response: closed-loop gain was higher for increases in neck tissue pressures than for decreases (Figure 6-6).

[5] Heart interval units are s^{-1}.

Mancia et al. mean regression coefficients ± standard deviations				
	MAP (mm Hg)		HI (ms)	
NTP	Early	Late	Early	Late
Increased	0.44 ± 0.13	0.65 ± 0.16	-3.4 ± 1.6	-2.7 ± 1.4
Decreased	0.55 ± 0.19	0.41 ± 0.13	-2.5 ± 3.3	-0.1 ± 1.5

Simulation regression coefficients from the CVSIM model				
	MAP (mm Hg)		HI (ms)	
s_0	Early	Late	Early	Late
Increased	0.54	0.72	−3.8	−3.6
Decreased	0.50	0.63	−4.9	−4.1

Table 6.3: Early (5–15 s) and late (90–120 s) responses in mean arterial pressure (MAP) and heart beat interval (HI) to step changes in neck tissue pressure. The first table is from 11 human subjects; the second is from a CVSIM simulation on the "normal" patient. Values are reported as linear regression coefficients. The human data, from Mancia et al. [35], is for calculated neck tissue pressure as partially transmitted from neck chamber pressure.

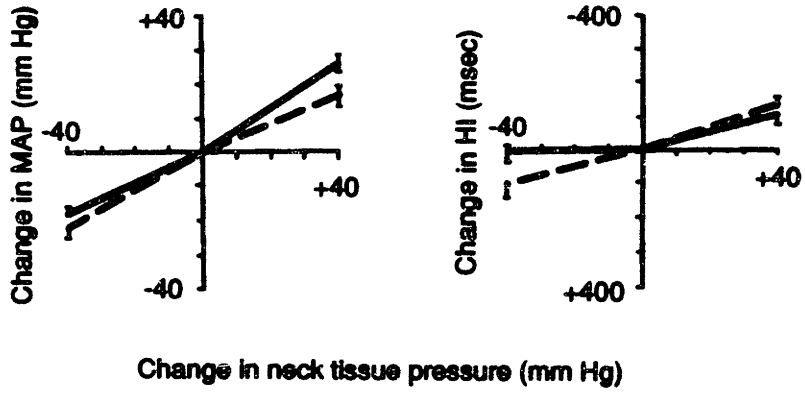


Figure 6-6: Changes in mean arterial pressure (MAP) and heart interval (HI) induced by changes in neck tissue pressure for a set of 11 human subjects. The dashed lines represent the early response, the continuous lines the steady-state response. See Table 6.3 for numerical values. The asymmetry between increased and decreased NTP in the late response is apparent. From Mancia et al. [35].

This trend is not as striking in the CVSIM simulation data.

Intra- and inter-individual variation in baroreceptor gains is large: the baroreflex sensitivity is dependent on several factors, including for example arousal state and general anesthesia [52]. In addition, trained athletes have reduced gains, and in fact, seemingly barely adequate control [58]. Considering the wide variability of baroreflex sensitivity, it is comforting to note that most of the gains during early (average of 5–15 s) and late (average of 90-120 s) response times in the CVSIM model are within one standard deviation of the 11-patient average from Mancia et al.The only major deviations occur in the late responses to decreased neck tissue pressure. As noted in Section 6.4.2, the slow time course of sympathetic withdrawal was not incorporated into this model; this may account for most of the difference in the late response.

6.6.2 Transient response to step inputs

An example of the transient response to changes in the arterial pressure setpoint s_0 is given in Figure 6-7; a human example for comparison with the same variables is given in Figure 6-8. A noticeable difference is the oscillatory ringing in the CVSIM response.

Baroreceptor setpoint changes and large hemodynamic parameter changes produce under-

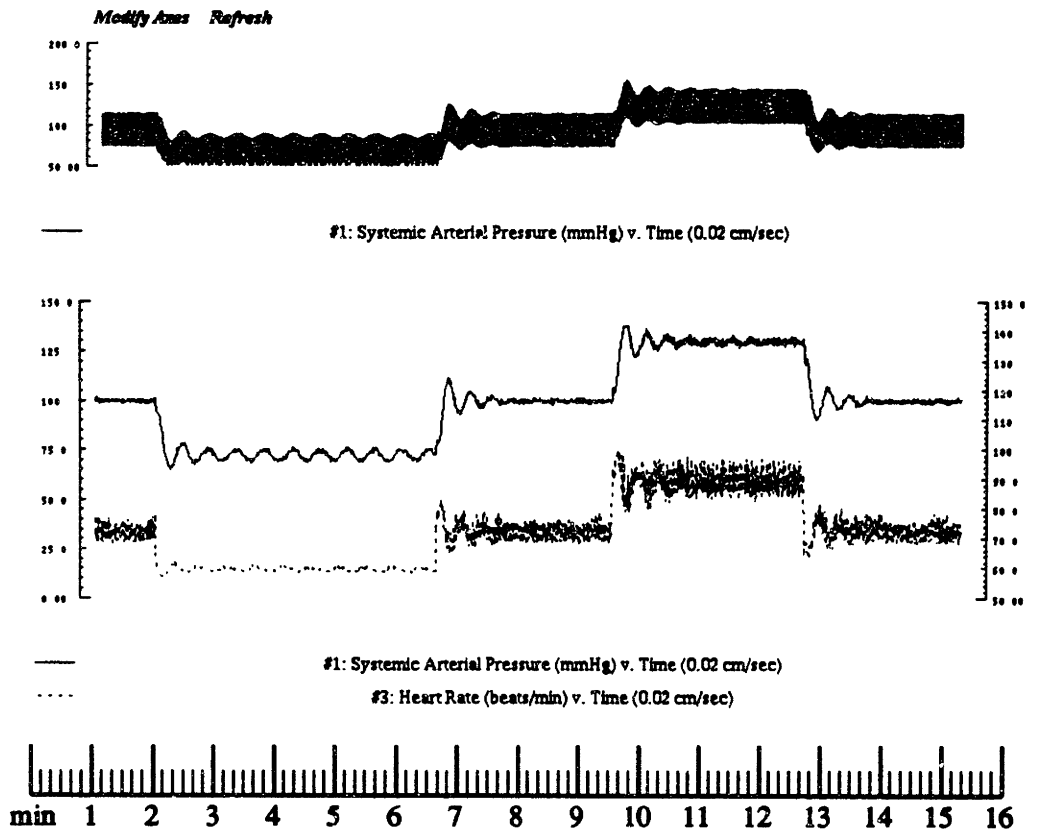


Figure 6-7: Simulation response to ± 40 mm Hg step changes in setpoint pressure. The time axis is 0.8 cm/min, or 75 s/cm, contrary to the label in the graph. Compare to Figure 6-8.

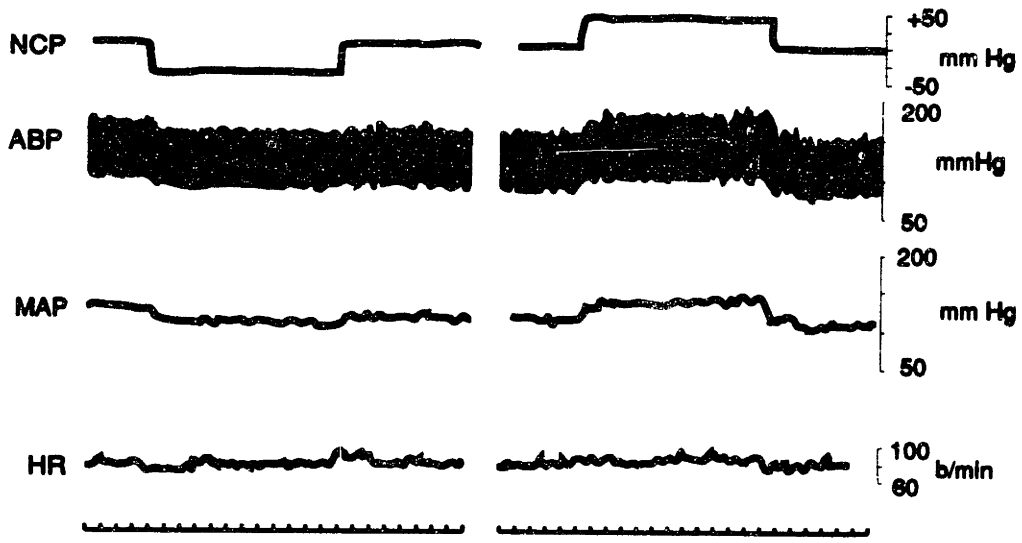


Figure 6-8: Hemodynamic changes induced by decrease and increase in neck tissue pressure in a human subject. NCP = neck chamber pressure; ABP = pulsatile arterial pressure; MAP = mean arterial pressure; HR = heart rate tachograph trace. Time is marked in 10 second intervals. From Mancia et al. [35].

damped responses, which can be minimized if the feedback gain is reduced. The oscillations are damped by an amount which is dependent on the current hemodynamic state. When the arterial pressure is low, sustained low-frequency oscillations can result (for example, see the first step deviation in Figure 6-7). Several investigators have reported such oscillatory behavior in compromised cardiovascular systems [56], [14], [18]. Madwed [34, p. 99] reported similar sustained oscillations in dogs after hemorrhage. Madwed's oscillations were at a frequency 0.05 Hz, whereas the oscillations in the CVSIM data are at 0.03 Hz.

6.6.3 Experimental stability

The underdamped responses to dynamic parameter manipulations in CVSIM are indicative of marginal stability of the system. However, the oscillations produced decrease in magnitude with each cycle unless the sympathetic gain is raised or other parameters significantly varied.

With the parasympathetic response intact and set to the normal value of 9 ms/mm Hg, the sympathetic gain must be raised from its normal of 9 ms/mm Hg to over 14 ms/mm Hg in order to produce sustained, nondecreasing oscillations. With the parasympathetic feedback gain set

to zero, as in pharmaceutical parasympathetic blockade, the sympathetic stability threshold is reduced to 13 ms/mm Hg. Values above 9 ms/mm Hg for sympathetic gain were observed to cause a dramatic increase in the time required for dissipation of oscillations.

Classical bounded-input/bounded-output instability is impossible for this system because of the feedback limiter ($\tan^{-1}$) and the stability of the separate linear components, as was described earlier.

Thus we see that the baroreflex control system model achieves the limited objective of a first-order approximation to the physiological system: the closed-loop gain, temporal responses, and oscillatory behavior are similar, but not identical, to those observed in humans.

Chapter 7

Cardiovascular Simulator Utilization

The success of this project is ultimately decided by those for whom the software was developed: students. This chapter describes the use to which CVSIM is placed and evaluates the simulator based on student participation and responses to questionnaires.

7.1 Usage in Engineering and Medical Courses

Engineering undergraduates, medical engineering graduate students, and M. D. candidates are exposed to the simulator early in the term during a unit on hemodynamics. They become familiar with the program through reading a user manual, attending a hands-on introductory laboratory, and performing a set of take-home problem sets. The M. D. course returns to the simulator later in the term for use as an adjunct to patient case studies, whereas the undergraduate course includes a student case presentation based directly on the simulator.

The user manual (included as Appendix A) was developed to introduce students to CVSIM and the Project Athena computing environment. The manual begins with a succinct description of the mathematical models associated with the simulator, followed by a description of Project Athena, and information on obtaining accounts and logging in to the system. A set of tutorial exercises guides the student through all the interaction windows of the user interface, and leads to an introductory hemodynamics problem. The final part of the user manual is a reference description of the user interface and system parameters.

A laboratory session is held to acquaint users with the system and to allow students to

work on their first simulator assignment in a supportive environment. One or two homework assignments use the simulator to explore specific hemodynamic issues. Students are required in some assignments to submit simulator printouts and to describe their findings, while in other assignments students may electively use the simulator to verify their understanding for a particular question.

In some semesters, course instructors have assigned presentations of patient cases to pairs of students. For these presentations, students study a case history, investigate the hemodynamic state of the associated patient simulation, and from this diagnose the patient's disease and assess hemodynamic compromise. They then prepare overhead graphs to demonstrate the important hemodynamic consequences of the patient's disease. Their methods and findings are presented to the class. The simulator has also been used in a take-home mid-term examination question on hemodynamics.

7.2 Utilization evaluation methods

The fall, 1989 group consisted of 15 M. I. T. undergraduates, predominantly upperclass engineers, of whom eight completed the questionnaire. The spring, 1990 group consisted of 46 Harvard and M. I. T. graduate and medical students, predominantly M. D. candidates, of whom 25 completed the questionnaire. All students used the simulator; however, the versions were different each semester. The fall and spring versions of CVSIM operated as described in the user manual (Appendix A), except that the ability to mark plots at points of interest with precise values was not introduced until the spring version.

7.2.1 Usage monitoring

Whenever the simulation was run from any Project Athena workstation, a log entry was recorded, including user name, date and time the simulation was started, and date and time the simulation was ended. These logs were used to calculate usage time for each student and for the class as a whole. At the conclusion of each simulation session, the user was prompted for general comments about the program and any difficulties encountered.

7.2.2 Questionnaires

In order to assess student opinion in a more objective manner, a questionnaire was developed consisting of agree-disagree statements and short-answer questions pertaining to the cardiovascular simulation and its incorporation into the course. The fall questionnaire contained 15 agree-disagree statements and 11 short-answer questions, whereas the spring questionnaire contained 18 agree-disagree statements, four short-answer questions, and space for additional comments.

For each statement, students were asked to rate their agreement on a scale from 1, for "Strongly Agree," to 5, for "Strongly Disagree." The statements were developed in paired sets, each pair asking about the same topic, one with a negative tone and the other with a positive tone. These statements were interspersed to arrange negative statements in proximity to positive statements, and with several unrelated statements separating any two statements in a matched pair. This careful construction of the questionnaire was done to allow more accurate reporting of opinion by eliminating some forms of reporting bias, and to provide a mechanism by which the consistency of each individual's answers can be tested [19].

Students were also asked to provide short answers to open-ended questions, and encouraged to conclude with any additional comments. At the end of the questionnaire, they were asked to optionally provide their names and login identification so that their answers might be correlated to their usage of the program and their performance on hemodynamics assignments.

7.3 Student—program interaction results

Students in the fall, 1989 class of *6.022J/2.792J/HST-542J* and the spring, 1990 class of *HST-090* provided test populations on which to evaluate the usage of the simulator as described above. No control group of simulator-deprived students was available, because the instructors felt the use of the simulator to be a necessary component of the subject. The fall term and spring term student populations, assignments, and questionnaires were different from one another, so they will be discussed separately.

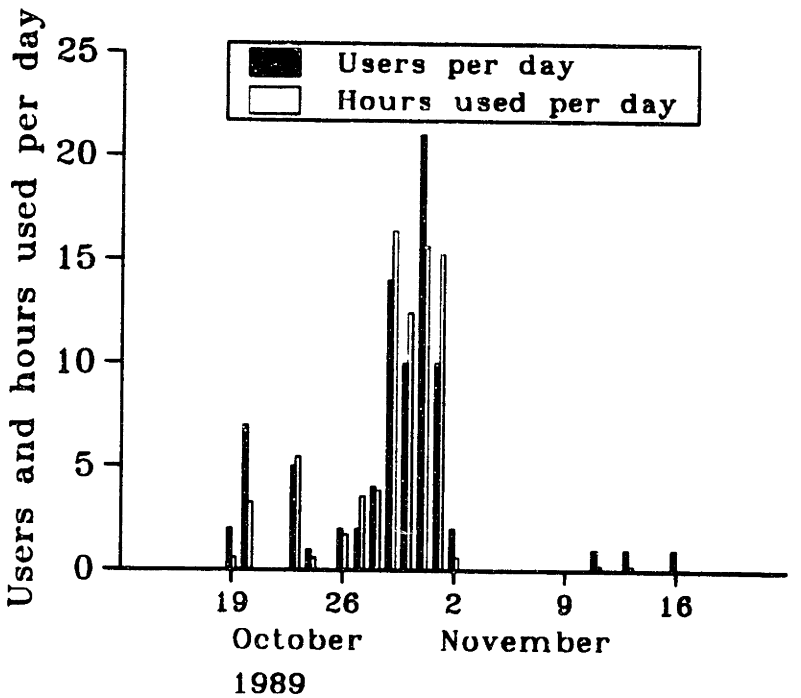


Figure 7-1: Number of logins and hours used per day, fall 1989, from a group of 15 undergraduate students. The large amount of activity in late October immediately preceded a pair of student presentation sessions based on CVSIM.

7.3.1 Program usage evaluation

Log messages were recorded every time the simulation was started and stopped. Because some students left their programs running and others killed the simulation before the end-time log could be written, some of the end-time logs are unavailable. As a reasonable substitute, the average usage time was calculated from available end-time logs and used in place of the recorded usage time for a missing 12% of the end logs in the fall group and 30% of the end logs in the spring group.

Figures 7-1 and 7-2 provide a comparison between the usage habits of the undergraduate and graduate/medical student populations. The undergraduate utilization of the simulator was more limited in the course calendar, but undergraduates individually spent more time per day on the simulator than the graduate/medical students. Both populations used the simulator in spurts which coincided with assignments; however, the background activity, outside of specific assignments,

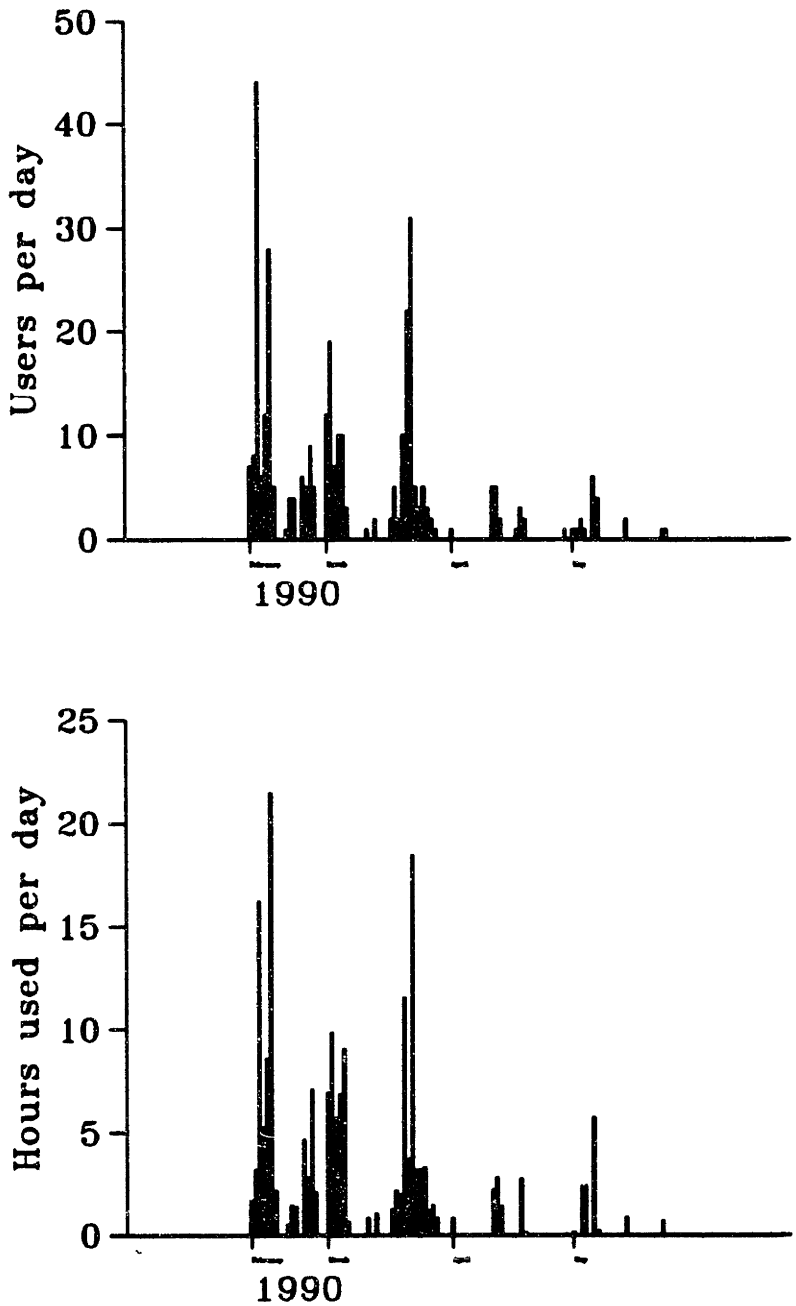


Figure 7-2: Number of logins and hours used per day, spring 1990, from a group of 45 graduate/medical students. Note that although "hours used" appears similar to Figure 7-1, the per-student usage is less by a factor of 3. Peaks are noted corresponding to an orientation laboratory Feb. 12, problem set due Mar. 5, home quiz due Mar. 22, and patient case discussions on Mar. 28, Apr. 18, and May 9.

appears to be greater for the graduate/medical student population. Because of the difference in amount of time spent per student per day, it appears that the graduate/medical students were able to make more efficient repeated use of CVSIM than the undergraduates.

7.3.2 Questionnaire results

Agree/Disagree statements

The agree/disagree statements are listed in Tables 7.1 and 7.2. The statements are ordered in matched pairs rather than in the questionnaire sequence. Means, standard deviations, and t-statistics were computed for each pair to determine whether that pair was answered significantly favorably or unfavorably.[1] A mean less than three on a first question, or greater than three for a second question in each pair corresponds to a favorable result. Negative differences correspond to a favorable balance between the two questions.

Although only eight questionnaires were obtained from the fall, 1989 group, four out of the seven statement pairs showed statistically significant differences; only one of those four had the difference in the unfavorable (> 0) direction. The students responded that the simulator project was worthwhile, that they enjoy exploring simulations, or at least do not avoid computers; that the software and hardware ran acceptably, or at least that the printer was not always broken; and that course software can make the material easier to understand. The students in the fall group did not find any utility in the simulator beyond the specific course assignments. They were more equivocal about whether CVSIM was boring or fun and whether such simulations should be developed for all their other subjects.

In contrast, the spring, 1990 group of 25 questionnaires from graduate and medical students had significant differences between every pair of answers, with only one pair with an unfavorable response. This group was adamant that the simulation was useful, fun and effective at demonstrating how hemodynamic variables depend on component properties ($p < 0.0005$). They also thought that they would not have learned as well without the simulator, that its use should be required, and that they used the simulator outside of the normal simulator assignments. However, they did not generally utilize the written documentation for CVSIM.

[1] Rosner [50, p. 243] is a good applied statistics text.

	STATEMENT 1: Agree – 5: Disagree (N = 8)	MEAN	STDEV	p VALUE
1.	The Cardiovascular Simulator project was a useful learning experience.	1.63	1.06	
5.	The project was a waste of time.	4.38	0.74	
	DIFFERENCE:	−2.75		< 0.005
3.	I wish all courses used software such as CVSIM.	3.63	0.92	
9.	I avoided using the software as much as possible.	3.50	1.41	
	DIFFERENCE:	0.13		NS
6.	I like exploring computer simulations.	3.25	1.04	
15.	I avoid computers whenever possible.	4.25	0.89	
	DIFFERENCE:	−1.00		< 0.1
8.	I had fun using the program.	3.00	1.07	
2.	Using the software was boring.	2.63	1.06	
	DIFFERENCE:	0.38		NS
10.	I used the simulator for other problem sets as well as the project.	5.00	0.00	
13.	The simulator was useful only for the project assignment.	3.50	1.20	
	DIFFERENCE:	1.50		< 0.005
12.	The software and hardware all ran smoothly.	3.00	0.93	
7.	The printer was always broken or out of paper.	4.25	1.16	
	DIFFERENCE:	−1.30		< 0.05
14.	The simulator was essential for me to understand cardiovascular hemodynamics.	3.75	1.16	
11.	No amount of course software would make the material easier to understand.	4.50	0.53	
	DIFFERENCE:	−0.80		< 0.1
4.	The project would have been better without the computer.	4.00	0.53	

Table 7.1: Fall, 1989 questionnaire statement responses. Statements are paired by category, with the positively-phrased question first in each pair; the original ordering of questions is listed to the left. A negative difference of means corresponds to a favorable rating. Significance was calculated from paired t-tests for the difference in response between the positive and negative question forms for each topic (NS = no significant difference).

	STATEMENT 1: Agree – 5: Disagree (N = 25)	MEAN	STDEV	p VALUE
1.	The cardiovascular simulator provided a useful learning experience.	1.56	0.71	
4.	The simulator assignments were a waste of time.	3.84	1.31	
	DIFFERENCE:	−2.30		< 0.0005
8.	I had fun using the program.	2.20	1.00	
2.	Using the software was boring.	3.84	1.07	
	DIFFERENCE:	−1.60		< 0.0005
3.	The use of the simulator should remain mandatory for everyone taking HST-090.	2.28	1.17	
14.	Although the simulation is fun for some people, its use should not be required formally in the course.	3.00	1.41	
	DIFFERENCE:	−0.60		< 0.15
5.	The simulator is effective in demonstrating how hemodynamic variables depend on component properties.	1.68	0.85	
16.	The simulator is not helpful for understanding complicated physiological interactions with many parameters.	3.40	1.53	
	DIFFERENCE:	−1.70		< 0.0005
11.	The control system model is an important adjunct to experiments on the uncontrolled system.	2.30	1.06	
7.	The control system model was difficult to use effectively.	3.55	1.18	
	DIFFERENCE:	−1.00		< 0.01
18.	I used the simulator electively to understand concepts or work on assignments outside of the explicit simulator assignments.	3.16	1.49	
9.	I avoided using the simulator as much as possible.	3.76	1.27	
	DIFFERENCE:	−0.60		< 0.15
13.	I would not have understood hemodynamics nearly as well without the cardiovascular simulator.	2.52	1.16	
10.	No amount of computer software would make the course content easier to understand.	4.12	1.27	
	DIFFERENCE:	−1.60		< 0.0005
6.	I like exploring computer simulations.	2.40	1.12	
15.	I avoid using computers whenever possible.	4.24	1.13	
	DIFFERENCE:	−1.80		< 0.0005
17.	The documentation for the simulator was helpful.	2.83	0.92	
12.	I did not utilize the CVSIM user manual very much.	2.08	1.26	
	DIFFERENCE:	0.64		< 0.1

Table 7.2: Spring, 1990 questionnaire statement responses. See caption of Table 7.1 for organization. A negative difference corresponds to a favorable rating.

There are many confounding factors that might have caused the differences between these populations of students: medical students may be less frank in the responses or have lower expectations about teaching software than M. I. T. undergraduates or the instructors or teaching assistants may have imparted different information regarding the simulator; however, there are several other possible reasons which deserve consideration. The graduate and medical students may have been more open to new teaching approaches, more direct in their approach to problems, and more mature in their evaluations than the undergraduates. It may be that a less intensive assignment mix over a longer period of time engenders a better feeling towards the simulator, or that the hardware or software actually performed better in the second term.

Note that these generalizations may not be accurate: pitting the disagreement of a negative question against the agreement to a positive question in no way guarantees validity of results. The results are dependent on the student's predilection to answer favorably, the emotional appeal of the statements, and other criteria which are difficult to identify. However, to improve dramatically on the validity of the study would be of little benefit compared to the subjective information available from free-form comments.

Written comments

Valid objective utility information is difficult to obtain and of dubious value compared with the wealth of subjective reactions in oral and written comments from students and course faculty. The short-answer questions from the questionnaires can supply much more information with less cost than a wholly objective approach.

Students were solicited for written comments through the questionnaires, through electronic mail, and as part of a laboratory writeup. A complete transcription of the fall, 1989 and spring, 1990 responses is given in Appendix B. For a totally unbiased view of the data, one should read the appendix itself. I will only attempt to provide my impressions of the student comments.

Fall 1989 comment impressions The undergraduates were asked for general impressions, descriptions of problems, descriptions of what was learned, and how the course and software might be improved. Most of the students' responses to general questions were favorable: most students mentioned "useful" as a general impression, although a few thought that it was difficult to become

familiar with the user interface. Most complained about the inability to show two different simulations simultaneously, although that was possible if the student ran two independent instances of CVSIM on his or her display. A few reported crashes, presumably due to programming bugs. Listed strengths of CVSIM included a good representation of how changes to the cardiovascular system affect the state of the system, of how computer model implementations can show the limitations of mathematical knowledge about a system, and more specifically it was strong in teaching about time-varying elastance theory, linear resistance, and compartmental capacitance models of physiology which are an integral part of CVSIM. Several students mentioned the need for a computer-based tutorial; evidently the introductory lab in that term was insufficient to acquaint the students with the computer program.

Spring 1990 comment impressions The Ph. D. and M. D. candidates were asked fewer written questions than the undergraduates; however, the questions covered the same general topics. The medical students enjoyed seeing the pressure – volume loop and the complex interactions between parameters and variables. Some said that the simulator provided a good framework for understanding physiology; one stated that most of what he or she learned about physiology was from the simulator. One of the assignments during the term required the calculation of stroke work from a P-V loop, which was not easy to determine; this engendered some negative feelings. The major problems for these students were slowness of the IBM RT/PC computers and problems waiting for printouts from the now-defunct IBM 3812 PagePrinter. Overall responses, however, were positive, and seemed to reflect an enjoyable learning process. There were very few recommendations for an on-line tutorial: the introductory laboratory may have been better managed, or possibly the medical students had less trouble mastering the user interface.

Chapter 8

Discussion

In this thesis I have described my efforts to improve and extend a computer teaching model of cardiovascular hemodynamics. I have provided verification of the model parameters from the physiology literature; developed, implemented, and verified a model of the baroreflex control system; designed and implemented an advanced real-time graphical user interface; and demonstrated the utility and acceptance of the resulting computer program in the teaching environment for which it was designed.

Although the objectives stated in Chapter 3 were met, there remain many possible improvements which would provide a more accurate and versatile simulation.

8.1 Possible model enhancements

A trade-off is inevitable between the elegant simplicity of a model and its performance in comparison with observed data. When modelling a complex physiological system for educational purposes, decisions must be made to restrict the scope of the model and to ignore certain physiological realities, in order to keep the system as intuitively understandable as possible. However, carefully chosen model enhancements can illuminate additional physiological principles and enrich the learning experience. The baroreflex control system is one example of the many extensions which can be made. I will give a brief overview here of some possibilities for further efforts in developing the hemodynamics and baroreflex teaching models, and will describe some simulation experiments which an improved model and interface could facilitate.

8.1.1 Hemodynamic model enhancements

Although an attempt was made to provide the best linear lumped-parameter model of hemodynamics, there are several areas in which the partially-nonlinear aspects of the physiology could be incorporated to the advantage of the student.

Compartmental volume limits

The current model has no upper bound on the volume of a compartment: volume increases linearly with increased pressure *ad absurdum*. This is especially problematic in the right ventricle, which is filled to its well-defined capacity in diastole in the normal regime of cardiac function.

It was necessary to decrease $C_{r,dias}$ and increase R_v in order to maintain appropriate levels of filling during normal operation; even so, the cardiac output responds inaccurately to increased heart rate because RV filling is overly dependent on filling time. If each compartment had a maximum volume "V_{max}" defined, beyond which no inflow were allowed, the diastolic ventricular compliances could be increased. In such a volume-limited simulation, heart rate would have a greater, and more physiologic, effect on cardiac output and arterial pressure than in the current model. Alternatively, a breakpoint in the pressure-volume relation could be defined, beyond which the incremental capacitance is greatly reduced. This alternate approach might allow a more accurate modelling of the approach to maximum volume.

The collapse of veins

It is also possible for volumes to become negative in the current simulation. This unsavory situation can develop when the external pressure on a vessel is increased to greater than its internal pressure. This could happen, for instance, during the Valsalva maneuver, when transthoracic pressure can reach +40 mm Hg, and the pulmonary vascular pressures were originally around 30 mm Hg.

Instead of collapsing, the veins "mathematically intussuscept" into their adjacent compartments, giving up blood volume that they do not have. This problem could be rectified by a nonlinearity to prohibit outflow from compartments with zero total volume.

Pulmonary resistance and the vascular waterfall

The pulmonary system poses special problems in modelling. The pulmonary microvasculature sits in a bed of varying external pressure due to gravity effects on the lung. As cardiac output increases, blood is forced higher into the lung tissues, and the number of parallel capillaries in use is increased. This gives rise to a decreased R_{pv} for increased flow, resulting in a more constant P_{pa}. Permutt and Riley [45] described this "vascular waterfall" mathematically. In order to model the pulmonary resistance adequately for detailed pulmonary hemodynamics experiments [38], [42], it will be necessary to make R_{pv} dependent on P_{pa} and P_{pv}. It may be enough to make the resistance a function of P_{pa} alone; this would reduce the required computation of resistance to the equivalent of a one-dimensional look-up table.

Nonlinear arterial system

The most visible and important time-invariant compartment in the simulation is the arterial system. There are three ways in which this system is modelled inaccurately to the extent that it is noticeable by students:

- The physiological arterial system is nonlinear over the range of pressures pertinent to the simulation.
- There is no model for the deceleration and rebound of blood at the end of ventricular ejection, and therefore no dicrotic notch.
- The arterial pressure is given only at the root of the aorta: transformation of the pressure pulse through the branching arterial tree is not modelled, whereas it is easily measured in humans.

The easiest addition would be a nonlinear pressure-volume relation for the arterial compartment (see Section 4.1.3, page 34). Li et al. [32] have proposed an iterative approach to estimating the nonlinear arterial compliance in man. For modelling specific patients, this would be a great improvement over data obtained from autopsy studies such as in Figure 4-1. However, we generally are not interested in modelling a specific subject, but in putting together the essential concepts which are the bases of the system's behavior. For this reason, it may be preferable to

approximate the arterial pressure-volume relation by a smooth curve which might be modifiable by the student to alternatively mimic young, relatively elastic arteries or elderly, atherosclerotic arteries. The most obvious result of such a nonlinear model would be that hypertension leads to larger pulse pressures.

In order to produce a dicrotic notch in the arterial pressure waveform, the inertance of blood must be modelled. This would require, in electrical circuit terms, the addition of an inductor element in series with ventricular outflow. Although the dicrotic notch is visible in arterial pressure tracings, it is not of overall significance to the circulation. It might be useful, however, to have an optional dicrotic-notch generator to demonstrate the origin of this ubiquitous arterial pressure feature.

The distributed properties of the arterial tree have not been addressed by the model. Many comprehensive arterial models have been developed, but most would require the addition of several parameters to the system, and would unnecessarily complicate the simulation for students. If the distributed properties are to be investigated, it should be done using a model independent from the current one.

It would be reasonable to attach an externally-defined model to CVSIM through a set of programmed routines; however, a description of how to do so is outside the scope of both this thesis and the courses for which CVSIM was designed.

The addition of atria

It might be useful to add atrial compartments to the simulation. This would require the addition of four elements: two time-varying capacitances and two resistances. Their primary utility would be in studying high-output states such as exercise, and possibly atrial stretch reflexes and other more sophisticated matters. Currently, computation time is enough of a concern to require the fewest number of parameters possible; atria were deemed to be insufficiently important to the simulation and were thus left out.

Left and right heart time-varying elastance

In the investigation of time-varying elastance of the ventricles, it was learned that the current C_l produces an $E(t)$ which does not agree closely with a published average $((1/P)\,dP/dt)_{max}$

of 44 s^{-1}. The current model reflects a relatively weak isovolumic contractility. In addition, the radically different geometries of the two ventricles lead to different $E(t)$ curves for left and right, which could be based in part on human RV elastances measured by Italia and Walsh [16]. Separately determined $E_l(t)$ and $E_r(t)$ with a careful, multiple-parameter fit for each ventricle would be an improvement over the provisional elastance curve in CVSIM.

Force-length-velocity modelling of the contracting ventricle

Our model assumes a time varying capacitance of the ventricle, in common with Sunagawa and Sagawa [60], as the basis for the driving force in the circulation. Cardiac contraction can be more carefully modelled as a set of functions relating maximal force to fiber length, and maximal velocity to fiber length. Beyar and Sideman have developed a model which incorporates the force-length-velocity model to determine the cardiac outflow [7]. However, for simplicity our model uses only the time-varying elastance and the inflow and outflow resistances to determine flow, even though it may lead to unrealistic fiber-shortening velocity and large forces on the heart muscle.

An implementation of the force-length-velocity model to determine outflow during ejection would provide a much more accurate model of the ventricle, especially in pathological situations. For example, in critical aortic stenosis the current model develops a very high pressure peak in early systole due to rapidly-rising $E(t)$; this anomaly could be prevented by a contraction-based model.

8.1.2 Reflex model enhancements

The baroreflex control system could be improved in several ways. As with the basic hemodynamic model, there is always a trade-off between simplicity and accuracy which must be considered in the context of teaching physiology.

Assymetric sympathetic response

As described earlier, sympathetic withdrawal has a much slower time course of action than sympathetic stimulation, because of the continued presence of neurotransmitter in the synapses after stimulation has ceased. One could use the current sympathetic impulse response for the

onset of sympathetic activity, and substitute another impulse response with a longer delay for sympathetic withdrawal. This would incorporate nonlinear behavior in the form of a switch, which would choose the fast response when s_0 is increasing, and the slow response when s_0 is decreasing. This would require that an additional signal history be kept, to store the action-or-withdrawal state of the sympathetic system. The implementation implications of such an asymmetric system have not been fully considered.

Gain versus oscillation

A more complete analysis of the delay and gain associated with the baroreflex, with particular emphasis on parameter sensitivity, should be conducted to evaluate the stability of the controlled CVSIM model as compared to physiologic studies. Sagawa's analyses might prove useful for this study [51].

Recently it was found by Clark [13] that the unmodified DeBoer model exhibits marked parameter sensitivity, an unstable oscillation pattern being easily demonstrated by small parameter deviations. A defect in that model is that the feedback time response is dependent on heart rate due to the beat-to-beat nature of the data. Although underdamped oscillations are observed, the current CVSIM model as implemented here does not show such marked parameter sensitivity. Small feedback gain modifications have proven effective in restoring behavior well within the limiting bounds of s' for all hemodynamic parameter settings which have been investigated.

Availability of control system parameters for modification

Currently, only the pressure setpoint s_0 and overall parasympathetic and sympathetic gains are available for user modification. In order to perform comprehensive experiments in cardiovascular control, the impulse responses, or at least the mean delay times, should be under user control. Additional user interface widgets are all that is needed to put the control system completely under the control of the experimenter.

Automated s_0 setpoint choice for nonstandard patients

When the control system is activated on a patient with a mean arterial pressure different from the expected 99.0 mm Hg, a baroreflex response is initiated to return the blood pressure to the

setpoint. Frequently, it would be preferable to automatically change the setpoint to the current mean arterial pressure; this would be a crude model of the well-known "long time constant" adaptability of the baroreceptor.

Additional hemodynamic reflexes

It is known that the arterial baroreceptors do not provide the only input signal to short-term cardiovascular regulation. The atrial stretch and other low-pressure venous and pulmonary baroreceptors probably have as great an influence on venous tone, for example, as do the arterial receptors, and they have a significant input to the heart rate [33], [26], [27]. A general, user-configurable control system could be developed which linearly controls any parameter based on any set of variables, with a graphically-specified impulse response. Such a "universal control" model might even be useful as a tool for original physiology research.

8.1.3 Possible extension for physiological experiments

With CVSIM as a basis for experimentation, it would be desirable for the experiments described in physiology textbooks to be easily accomplished on the simulation. Some things, such as Guyton's cardiac output/venous return curves, are difficult but possible to accomplish with the current CVSIM; others require model or user interface extensions to be performed.

Altered network topology: shunts

Many students have asked about the possibility of producing shunts of various kinds in the simulation. Currently, this is not possible: each additional connection would add terms to two of the state equations, which are not modifiable except at program compilation. If students were to write their own set of simulation equations and compile their own programs, however, almost any network could be realized. This would change the nature of the student interaction, allowing much more flexibility and a deeper appreciation for modelling constraints. It seems infeasible at present to add topology flexibility to the run-time computer simulation: too much computational overhead would be involved in checking all the possibilities.

There are a few simple topology enhancements which would cover most of the physiologically relevant flow anomalies and which could be added more easily: valvular regurgitation could be

modelled with leakage resistances in parallel with the diodes, and a possible ventricular shunt could be added by connecting the ventricles by a resistance which is normally very high.

Open-system experiments

Many of the classical physiology experiments involve opening the cardiovascular system of an animal and controlling all parameters except those of interest; for example, the Starling heart–lung preparation. It would probably be easier to develop separate sets of state equations for each such experiment, and then provide a menu of situations from which to choose a simulation.

Cardiac output/venous return curves

One of the more useful concepts in understanding the interaction between the heart and the vascular system is the cardiac output/venous return curve. Guyton developed this approach [24], which displays a curve of cardiac output given a consistent heart under varying loads, and a curve of venous return given a consistent vascular system under varying cardiac pumping strength. The intersection of the two curves is the normal operating point. Interventions can be described as moving either the cardiac output curve or the venous return curve, or both.

It is fairly difficult to produce this graph with CVSIM (Figure 8-1), because it requires connecting operating points from multiple steady-state experiments. A user interface object which allows the collection of data points and then plots them on a graph would make such experiments much easier.

8.2 Possible implementation enhancements

In addition to enhancements of the underlying mathematical models, there are several interesting possibilities for improving the implementation and user interface.

8.2.1 Clocked output for real-time operation

When the simulation was first being developed as a dynamic simulation, the workstation platforms were slow enough to make actual real-time simulation infeasible. However, the past three years has seen an incredible increase in the processor speed of engineering workstations: the simulation

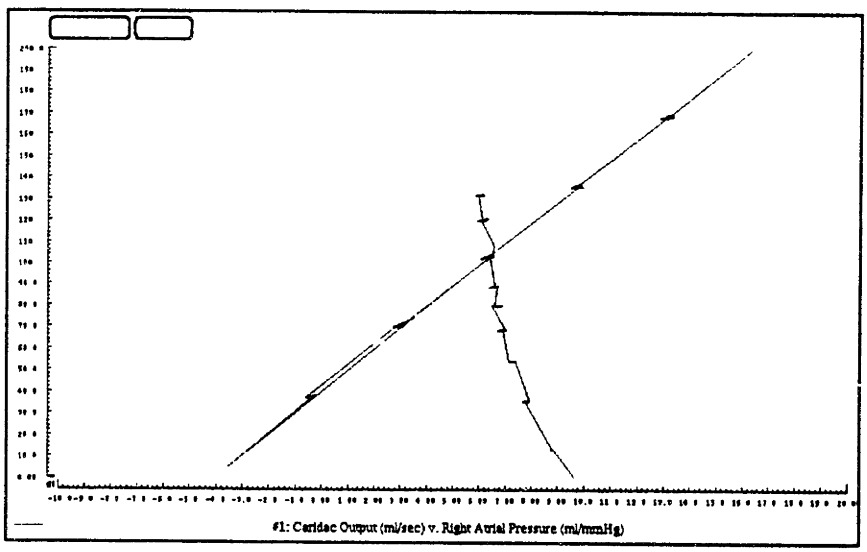


Figure 8-1: Guyton's cardiac output/venous return curve reproduced from CVSIM. The cardiac output curve was obtained by varying total blood volume in 500 ml increments from 3500 ml to 6000 ml. The venous return curve was obtained by varying cardiac diastolic compliances from systolic values to twice the normal diastolic values; this changes the stroke volume, so that cardiac output varies. The irregularities in the venous return curve are the result of changing the right and left heart compliances unevenly as the values shift between a ratio of 2:1 for large capacitances and 3:1 near the systolic values.

now runs on a DECstation 3100 at up to 4.5 times real time! With the expectation that hardware capabilities will only increase, it would be reasonable to tie the simulation time to real time using the system clock, and to allow other processing, such as a tandem simulation, to occur with the remaining clock cycles. This would not be an unreasonable addition in practice; it simply has not been done yet. An additional benefit to a clocked simulation would be an improvement in the response time for mouse button clicks, since the X instruction queues will not be filled at the maximum possible rate.

8.2.2 Multiple patients

One of the most common complaints about the simulator is that it presents only one patient at a time. With increased processor speed, it would be easy to adapt the user interface to control multiple distinct simulations. A difficult part of this addition would be a mechanism for distinguishing the simulations graphically; they could be numbered or named, but space for the simulation name or number must be created for all of the plot and interaction windows. If color workstations were available, it would be a simple matter to give each patient a different color scheme.

8.2.3 User storage and retrieval of simulation state and images

After a long simulation run, it might be desirable to go back and look at variables which were not originally plotted; saving portions of the simulation data on the student's disk space would be even better. It is not currently done because of the overhead involved in moving large amounts of data during the simulation. Perhaps an option to "capture" simulation data to disk at appropriate times in the experiment would maintain normal simulation efficiency while supplying the needed functionality.

Likewise, images of data should be saveable. Use of X commands in Unix allows complete freedom of storage, manipulation, formatting, and printing of the X windows from the simulator, but this is not documented in CVSIM and does not have the point-and-click simplicity of the built-in screen dumps. A few buttons could be added to the simulator for loading and dumping, as well as printing, X windows.

8.2.4 Automated interventions

Some students have been disappointed with the lack of "realistic" physiological state and intervention choices in CVSIM. These comments have usually been phrased in terms of drug action; for example, "Why can't I just give the patient some epinephrine and watch what happens?" Instead, the student must estimate the actions of epinephrine on the circulation, and modify the relevant parameters. Some of the course faculty have praised this requirement for rigorous understanding of drug action; however, most students will not try complex parameter manipulations because of the difficulties involved in arriving at a "correct" simulation. The addition of a menu of drugs, which would modify the system appropriately according to a time course of action, would enhance the clinical flavor of the student interaction, especially for the patient case studies.

In addition, it may be useful to allow for arbitrary time-varying parameters. For instance, P_{th} and R_p are known to vary with breathing. Studies of baroreflex control and other simulations with long time constants would be made more realistic by such variations.

8.2.5 Spectra of variables

The study of cardiovascular control relies heavily on frequency-domain methods. A widget which plots the Fourier transform of Kaiser-windowed heart rate or arterial pressure, for instance, would allow for some interesting comparisons to the cardiovascular variability literature.

8.2.6 Improved printouts

Currently, the printouts obtained from the simulator are simply X screen dumps produced by the `xdpr` command. Much higher-quality graphics can be produced from the PostScript printers by constructing PostScript code directly from the plot data rather than printing the grainy-appearing screen windows. This would require the saving of plot data, but data storage is desirable anyway for user-specified redrawing of plots.

8.2.7 A dynamic simulation graphics library

The ultimate in flexibility for simulation would be for the student, or researcher, to write his or her own mathematical simulation routines, and to provide hooks for the user interface to manipulate

the simulation. The major problem is the definition of a flexible simulation-to-interface protocol. One might imagine a table of parameters, their names and limits, etc. as in `parameter.db` (see Section C.3, page 201), with predefined functions to be called at specified times, such as at startup, when requesting new data, or when a menu item is selected by the user.

In addition, the more versatile sections of code, principally the Plot and NumberLine widgets, could be used by any other X program requiring dynamic parameter manipulation and plotting. The X Toolkit protocol makes it very easy for an application programmer to "drop in" the widgets of his choice, including those created for this dynamic simulation.

8.3 Conclusion

The cardiovascular simulation CVSIM is itself a dynamic entity, and will never be complete. Its utility in teaching the basics of hemodynamics is unquestionable, but equally unquestionable are current deficiencies and possible improvements. As technology and physiologic knowledge continue to advance, there will always be room for additions and corrections to the cardiovascular simulator. I hope that its present incarnation will survive long enough to touch the lives of many future doctors and researchers.

Bibliography

[1] Appel, Marvin. *Parameter Identification in the Cardiovascular System.* PhD thesis, Harvard University, Cambridge, Massachusetts, 1991.

[2] Beeuwkes, R. and N. Braslow. The Cardiovascular Analog Trainer: A real time physiological simulator. *Physiology Teacher*, 3(3):4–7, 1974.

[3] Beneken, Jan E. W. A physical approach to hemodynamic aspects of the human cardiovascular system. In Reeve, E. B. and Arthur C. Guyton, editors, *Physical Bases of Circulatory Transport: Regulation and Exchange*, pages 1–46. W. B. Saunders, 1967.

[4] Beneken, Jan E. W. Some computer models in cardiovascular research. In Bergel, D. H., editor, *Cardiovascular Fluid Dynamics I*, volume 1, chapter 6, pages 173–223. Academic Press, London, 1972.

[5] Berger, Ronald D., J. Philip Saul, and Richard J. Cohen. Transfer function analysis of autonomic regulation I. Canine atrial rate response. *American Journal of Physiology*, 256(Heart Circulation Physiology 25):H142–H152, 1989.

[6] Berne, Robert M. and Matthew N. Levy. *Cardiovascular Physiology.* Mosby, St. Louis, fourth edition, 1981.

[7] Beyar, R. and S. Sideman. Model for left ventricular contraction combining the force length velocity relationship with the time varying elastance theory. *Biophysical Journal*, 45:1167–1177, June 1984.

[8] Bhatty, Dhimat-Apurva. Computer simulation of the human cardiovascular macrocirculation. SB thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, April 1987.

[9] Borst, Cornelius and John M. Karemaker. Time delays in the human baroreceptor reflex. *Journal of the Autonomic Nervous System*, 9:399–409, 1983.

[10] Brecher, Gerhard A. and Pierre M. Galletti. Functional anatomy of cardiac pumping. In *Handbook of Physiology, Section 2: Circulation*, volume II, chapter 23, pages 759–798. American Physiological Society, first edition, 1963.

[11] Burton, A. C. *Physiology and Biophysics of the Circulation*. Year Book, Chicago, second edition, 1972.

[12] Campbell, Kenneth, Marie Zegien, Thomas Kagehiro, and Harriet Rigas. A pulsatile cardiovascular computer model for teaching heart-blood vessel interaction. *Physiologist*, 25(3):155–162, 1982.

[13] Clark, Kevin. DeBoer model parameter sensitivity. Personal communication, May 1989.

[14] DeBoer, R. W., J. M. Karemaker, and J. Strackee. Hemodynamic fluctuations and baroreflex sensitivity in humans: A beat-to-beat model. *American Journal of Physiology*, 253(Heart Circulation Physiology 22):H680–H689, 1987.

[15] Defares, J. G., J. J. Osborne, and Hiroshi H. Hara. Theoretical synthesis of the cardiovascular system. Study I: The controlled system. *Acta Physiol. Pharm. Neerl.*, 12:189–265, 1963.

[16] Dell'Italia, Louis J. and Richard A. Walsh. Application of a time varying elastance model to right ventricular performance in man. *Cardiovascular Research*, 22:864–874, 1988.

[17] Dickinson, C. J., D. Ingram, and P. Shephard. A digital computer model for teaching the principles of systemic hæmodynamics ('MacMan'). *Journal of Physiology (London)*, 216:9P–10P, 1971.

[18] Downing, S. Evans. Baroreceptor regulation of the heart. In Berne, Robert M., editor, *Handbook of Physiology, Section 2: The Cardiovascular System*, volume I, chapter 17, pages 621–. American Physiological Society, Bethesda, Maryland, second edition, 1979.

[19] Gleitman, Henry. *Psychology*. Norton, 1981.

[20] Grodins, Fred S. Integrative cardiovascular physiology: A mathematical synthesis of cardiac and blood vessel hemodynamics. *The Quarterly Review of Biology*, 34(2):93–116, June 1959.

[21] Grossman, William. *Cardiac Catheterization and Angiography*. Lea and Febiger, third edition, 1986.

[22] Grossman, William. Pressure measurement. In *Cardiac Catheterization and Angiography*, chapter 9. Lea and Febiger, third edition, 1986.

[23] Guyton, Arthur C. Venous return. In *Handbook of Physiology, Section 2: Circulation*, volume II, chapter 32, pages 1099–1134. American Physiological Society, first edition, 1963.

[24] Guyton, Arthur C. *Textbook of Medical Physiology*. Saunders, Philadelphia, seventh edition, 1986.

[25] Guyton, Arthur C., Thomas G. Coleman, and H. J. Granger. Circulation: Overall regulation. *Annual Review of Physiology*, 34(13), 1972.

[26] Hainsworth, R., P. N. McWilliam, and D. A. S. G. Mary. *Cardiogenic Reflexes*. Oxford, 1987.

[27] Hainsworth, Roger. Vascular capacitance: Its control and importance. *Rev. Physiol. Biochem. Pharmacol.*, 105:101–173, 1986.

[28] Hallock, P. and I. C. Benson. *Journal of Clinical Investigation*, 16:595, 1937.

[29] Katz, S., R. G. Hollingsworth, J. G. Blackburn, and H. T. Carter. Computer simulation in the physiology student laboratory. *Physiologist*, 21(6):41–44, 1978.

[30] Laurens, Paul. Catheterization by an intracardiac micromanometer. In Zimmerman, Henry A., editor, *Intravascular Catheterization*, chapter 8, pages 514–544. Charles C. Thomas, second edition, 1968.

[31] Lawson, C. The volume of blood: A critical examination of methods for its measurement. In *Handbook of Physiology, Section 2: Circulation*, volume I, chapter 3. American Physiological Society, first edition, 1962.

[32] Li, John K.-J., Ting Cui, and Gary M. Drzewiecki. A nonlinear model of the arterial system incorporating a pressure-dependent compliance. *IEEE Transactions of Biomedical Engineering*, 37(7):673–678, July 1990.

[33] Linden, R. J. and C. T. Kappagoda. *Atrial Receptors*, volume 39 of *Monographs of the Physiological Society*. Cambridge, 1982.

[34] Madwed, Jeffrey B. *Dynamic Analysis of Arterial Blood Pressure and Heart Rate During Baseline Conditions and Hemorrhage in the Conscious Dog*. PhD thesis, Harvard University, Cambridge, Massachusetts, March 1987.

[35] Mancia, Giuseppe, Alberto Ferrari, Luisa Gregorini, Romano Valentini, John Ludbrook, and Alberto Zanchetti. Circulatory reflexes from carotid and extracarotid baroreceptor areas in man. *Circulation Research*, 41(3):309–315, 1977.

[36] Mark, Allyn L. and Giuseppe Mancia. Cardiopulmonary baroreflexes in humans. In Berne, Robert M., editor, *Handbook of Physiology, Section 2: The Cardiovascular System*, volume III, chapter 21, pages 795–813. American Physiological Society, Bethesda, Maryland, second edition, 1983.

[37] McKay et al, R. G. Left ventricular pressure-volume diagrams and end-systolic pressure-volume relations in human beings. *Journal of the American College of Cardiology*, 3:301, 1984.

[38] Milnor, William R. Pulmonary hemodynamics. In Bergel, D. H., editor, *Cardiovascular Fluid Dynamics*, volume 2, chapter 18, pages 299–340. Academic Press, London, 1972.

[39] Milnor, William R. Normal cuirciulatory function. In Mountcastle, Vernon B., editor, *Medical Physiology*, volume 2, chapter 37, pages 930–943. Mosby, thirteenth edition, 1974.

[40] Milnor, William R. The pulmonary circulation. In Mountcastle, Vernon B., editor, *Medical Physiology*, volume 2, chapter 42. Mosby, thirteenth edition, 1974.

[41] Mirsky, Israel. Elastic properties of the myocardium: A quantitative approach with physiological and clinical applications. In Berne, Robert M., editor, *Handbook of Physiology,*

Section 2: The Cardiovascular System, volume I, chapter 14, pages 497–531. American Physiological Society, Bethesda, Maryland, second edition, 1979.

[42] Mitzner, W. and I. Huang. Interpretation of pressure-flow curves in the pulmonary vascular bed. *The Pulmonary Circulation in Health and Disease*, pages 215–230, 1987.

[43] Noordergraaf, Abraham. Control. In *Circulatory System Dynamics*, chapter 8, pages 250–277. Academic Press, 1978.

[44] Noordergraaf, Abraham, Gerard N. Jager, and Nicolaas Westerhof. *Circulatory Analog Computers*. North-Holland, 1963.

[45] Permutt, S. and R. L. Riley. Hemodynamics of collapsible vessels with tone: The vascular waterfall. *Journal of Applied Physiology*, 18(5):924–932, 1963.

[46] Peterson, Kirk L., David Skloven, Philip Ludbrook, John B. Uther, and Jr. John Ross. Comparison of isovolumic and ejection phase indices of myocardial performance in man. *Circulation*, 49:1088–1101, June 1974.

[47] Peterson, Nils and Diana Armstrong. *Cardiovascular Systems and Dynamics*. Command Applied Technology, PO Box 511 Pullman, Washington 00163, 1985.

[48] Press, William H., Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. Adaptive stepsize control for Runge-Kutta. In *Numerical Recipes in C*, chapter 15.2, pages 574–579. Cambridge, 1988.

[49] Press, William H., Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. Runge-Kutta method. In *Numerical Recipes in C*, chapter 15.1, pages 569–574. Cambridge, 1988.

[50] Rosner, Bernard. *Fundamentals of Biostatistics*. Duxbury Press, Boston, second edition, 1986.

[51] Sagawa, Kiichi. The use of control theory and systems analysis in cardiovascular dynamics. In Bergel, D. H., editor, *Cardiovascular Fluid Dynamics*, volume 1, chapter 5, pages 115–171. Academic Press, New York, 1972.

[52] Sagawa, Kiichi. Baroreflex control of systemic arterial pressure and vascular bed. In Berne, Robert M., editor, *Handbook of Physiology, Section 2: The Cardiovascular System*, volume III, chapter 14, pages 453–496. American Physiological Society, Bethesda, Maryland, second edition, 1983.

[53] Sagawa, Kiichi, Hiroyuki Suga, A. A. Shoukas, and K. M. Bakalar. End-systolic pressure-volume ratio: New index of ventricular contractility. *American Journal of Cardiology*, 40(748), 1977.

[54] Sah, Robert and George Moody. *User Manual for the Cardiovascular Simulator*, 1985.

[55] Sato, Toshiro, Stanley M. Yamashiro, Daniel Vega, and Fred S. Grodins. Parameter sensitivity analysis of a network model of systemic circulatory mechanics. *Annals of Biomedical Engineering*, 2:289–306, 1974.

[56] Shepherd, John T. and Guiseppe Mancia. Reflex control of the human cardiovascular system. *Rev. Physiol. Biochem. Pharmacol.*, 105:1–99, 1986.

[57] Sjöstrand, Torgny. Blood volume. In *Handbook of Physiology, Section 2: Circulation*, volume I, chapter 4, pages 51–62. American Physiological Society, first edition, 1962.

[58] Stegemann, J., A. Busert, and D. Brock. Influence of fitness on the blood pressure control system in man. *Aerospace Medicine*, 45(1):45–48, 1974.

[59] Suga, Hiroyuki and Kiichi Sagawa. Models of circulatory mechanics. *Circulation Research*, 35(117), 1974.

[60] Sunagawa, Kenji and Kiichi Sagawa. Models of ventricular contraction based on time-varying elastance. *Critical Reviews in Biomedical Engineering*, 7(3):193–228, February 1982.

Appendix A

User Manual for the Cardiovascular Simulator

CARDIOVASCULAR SIMULATOR

User Manual
and
Software Reference Manual

Harvard-M.I.T. Division of Health Sciences and Technology
June 17, 1990

CONTENTS

Part I: User Manual

The simulator has been designed to be as user-friendly and self-explanatory as possible. Part I of this document, the *User Manual*, will serve as an introduction and as a description of the program's major features. Examples and student exercises round out this introductory material. The user is strongly encouraged to read through Part I, performing the examples and exercises to become better acquainted with all aspects of the program. Part II, the *Software Reference Manual*, describes every element of the user interface in an organized fashion.

1. Introduction

Purpose

The Cardiovascular Simulator is a dynamic computer simulation of human cardiovascular hemodynamics, primarily intended for students of physiology and medicine. It is implemented on engineering workstations running the X Window System, allowing the student to perform a variety of measurements in a real-time simulated cardiovascular system, not all of which would be possible in an animal laboratory. The simulation is based on the lumped-parameter mathematical model taught in Quantitative Physiology and Cardiovascular Pathophysiology courses in the Harvard-M. I. T. Division of Health Sciences and Technology.

Project Athena

The computer simulation is implemented on the M. I. T. Project Athena student computing network, composed of Unix workstations donated by I. B. M. and Digital Equipment Corporation. These workstations are connected on a campus network, and utilize the X11 window system. Project Athena staff are available for consultation and to report problems. Many other scientific and word-processing packages are available on Project Athena; see *Getting Started on Athena* for more information.

2. The Model

The Basic System

The model which forms the basis of the simulator, based on an analog computer implementation by Defares [1], is shown in Figure 1. The cardiovascular system is divided into four major sections: the left heart, the systemic (peripheral) circulation, the right heart, and the pulmonary circulation. Each side of the heart is modelled by a variable capacitor, representing the pumping action of atrium and ventricle as a unit, and two diodes, representing the atrioventricular and arterial valves, and inflow and outflow resistances. In total, the blood storage capacity of all vessels are lumped into six capacitances (C_l, C_a, C_v, C_r, C_{pa}, C_{pv}), connected by six resistances (R_{li}, R_{lo}, R_a, R_v, R_{ro}, R_{pa}), and four ideal flow valves (D_{li}, D_{lo}, D_{ri}, D_{ro}).

The program calculates experimental data based on constituent relations for each element and the continuity of flow through the system, analogous to Kirchoff's Current Law for electrical circuits. For the resistive pathways (resistors), flow is the pressure drop across the pathway divided by the resistance of the pathway, equivalent to Ohm's Law for linear resistors.

$$Q_r = P_r/R \qquad \text{Electrical Analog:} \quad i = v/r \tag{1}$$

For the capacitative compartments (capacitors), flow is equal to the time rate of change of volume in the compartment.

$$Q_c = dV_c/dt \qquad i = dq/dt \tag{2}$$

The volume in the compartment can be expressed as the pressure times the compliance, plus the the zero-pressure (atmospheric) filling volume.

$$V_c = P_cC + V_0 \qquad q = vc + q_0 \tag{3}$$

Therefore, the change in volume is

$$dV_c/dt = d(P_cC)/dt \qquad dq/dt = d(vc)/dt \tag{4}$$

This gives a complete equation for the continuity of flow:

$$Q_c = C\, dP_c/dt + P_c\, dC/dt \qquad i = c\, dv/dt + v\, dc/dt \tag{5}$$

The lumped-parameter model is described fully by Eq. (5) applied to each of six circuit nodes, with initial conditions. These equations are solved numerically, giving pressure, volume, and flow through each compartment of the system. Initial parameters are chosen which closely match experimental data from the literature. The variable capacitances in the heart are driven by a pre-computed time function, also derived from experimental data. The user of the simulator may modify all local and system parameters at any time; the instantaneous response to such perturbations can then be visualized for any pressure, volume, or flow at any point in the system.

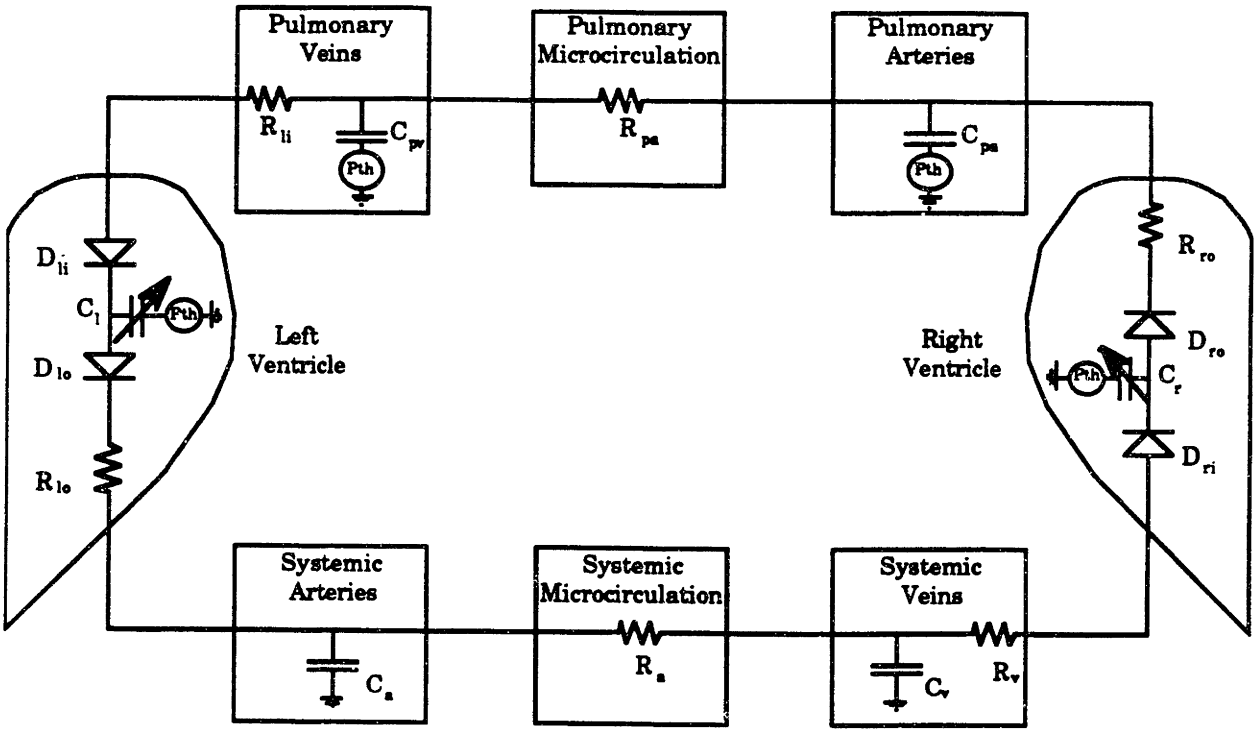


Figure 1. Circuit Diagram Equivalent of Lumped Parameter Model

The Baroreceptor Control System

The simulator incorporates a baroreceptor feedback control model in addition to the hemodynamics described above. The baroreceptor senses arterial pressure, P_a, producing an afferent signal, S, proportional to the deviation of P_a from the current setpoint pressure, P_{sp}. The physiological control pathways are described in Figure 2.

This control signal is used to modulate autonomic activity to control heart rate and contractility, peripheral resistance, and venous tone. We have chosen a proportional control law which updates HR, $C_{l\,sys}$, $C_{r\,sys}$, R_a, and V_{v0} by adding to each of these parameters a deviation consisting of a moving average of the control signal multiplied by a gain factor. The model is based on the work of DeBoer [2]. Each limb of the response has a specific delay due primarily to the response times of different autonomic nerve types and end-organs. Afferent nerve and central nervous system delays also contribute to delay in the control system (Table 1).

NERVE TYPE	LATENCY	MAX RESPONSE	LENGTH OF RESPONSE
Alpha-sympathetic	2.0 sec	5.0 sec	30.0 sec
Beta-sympathetic	2.0 sec	5.0 sec	30.0 sec
Parasympathetic	0.0 sec	0.5 sec	1.0 sec

Table 1. Delay values for reflex responses

Eq. (6) is the control equation for peripheral resistance. It incorporates a resistance setpoint R_0, an alpha sympathetic tone constant τ (usually zero), a gain factor γ, and a moving average of the control signal, S[n].

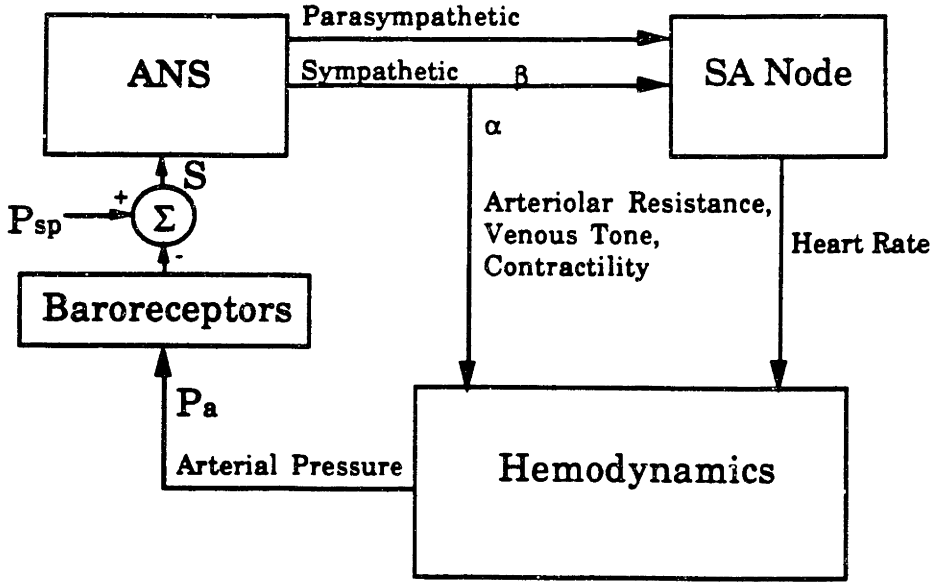


Figure 2. Baroreceptor Reflex Loop

$$R_a[n] = R_0 + \tau + \gamma \sum_{k=0}^{L} h[k]\, S[n-k] \qquad (6)$$

The other controlled parameters are set similarly, but with different values for τ, γ, the weighting function h[k], and the nominal parameter value setpoint.

The gain values for heart rate and peripheral resistance were obtained from DeBoer [2], whereas contractility and resistance gains were calculated directly from known near-maximal physiological responses in order to produce reasonable activity. This gross approximation may not hold for such a linearized operating-point control model; therefore caution should be used when making physiological inferences based on the control system model. For that matter, it should always be remembered that the cardiovascular simulator is not by any means a complete model of hemodynamics — it merely serves as a linear simplification for utility in understanding the principles of the real, nonlinear system.

REFLEX LIMB	GAIN	RESPONSE TIMING
heart rate	18 ms/mmHg	beta-sympathetic & parasympathetic
contractility	0.007 ml/mmHg/mmHg	beta-sympathetic
peripheral resistance	0.011 PRU/mmHg	alpha-sympathetic
venous zero-pressure volume	26.5 ml/mmHg	alpha-sympathetic

Table 2. Gain and timing for reflex responses

3. The Project Athena Computing Environment

The cardiovascular simulator was developed for Project Athena, the student computing environment at M. I. T. Project Athena combines the Unix operating system on individual computer workstations with network-accessible resources such as printers, software packages, and user accounts. While it is not necessary to know Unix to run the simulator, the student must be able to locate and activate a workstation, obtain a user account, and manipulate windows in the X window system in order to use the software effectively.

Project Athena provides a rich computing environment. In addition to the simulator and software for other courses, a wide variety of scientific, mathematical, and programming tools, word processing, electronic mail, and worldwide network access are available to the interested student: see *Getting Started on Athena* and other Project Athena documents for more information.

The Project Athena workstations

Each Athena workstation is an individual computer which relies on the network for centralized file storage and user authentication. For instance, we maintain only one copy of the cardiovascular simulator, which may be accessed simultaneously by several workstations.

HST and Whitaker College share a private workstation cluster in E25-131, which will serve as the teaching cluster for this course. Currently there are eight IBM RT/PC workstations, two DEC 3100 workstations, and a PostScript printer in the HST cluster connected to the Athena network. There are also many other public computing clusters across campus where this software may be accessed.

How to register for an account

Find an unused, deactivated workstation which has the "Hit any key to start" message or login window displayed. Hit a key and wait a minute if you do not already see the login window containing the Athena logo.

If you have never used your Athena account, move the mouse so that the pointer is on the small words "To register for a new account" and click the left mouse button. A registration window should appear. Make sure the mouse pointer is inside this window, and type in the requested information, hitting return when you are done with each question. If all goes well, you should be able to log in within 24 hours.

N.B. All students must register for an account BEFORE THE INTRODUCTORY LAB SESSION.

How to interact with X

All Project Athena workstations use the X11 Window System for user interaction. X manages your display screen, keyboard, and mouse as a collection of independent *windows*, each of which may represent a single computer program (Figure 3). The *window manager* allows you to move and resize these windows, and to choose a window for input by placing the mouse pointer inside the borders of that window. Each window is like a separate computer terminal, allowing you to do many different things at once. One such program is the vt100 terminal emulator, called xterm. To have another terminal on

your screen, just type **xterm &** (the ampersand starts the job in the background). The window manager will inform you that a new window has been created by changing the shape of the mouse pointer. Put the mouse pointer in the upper-left corner of where you want the window and click the left mouse button. Now you have two computer terminals. You must be careful to keep the mouse pointer inside the window of interest, because each window will only accept input when the mouse pointer is inside its borders. In this way, you can type a letter to a friend in one window while compiling a program in another, for example.

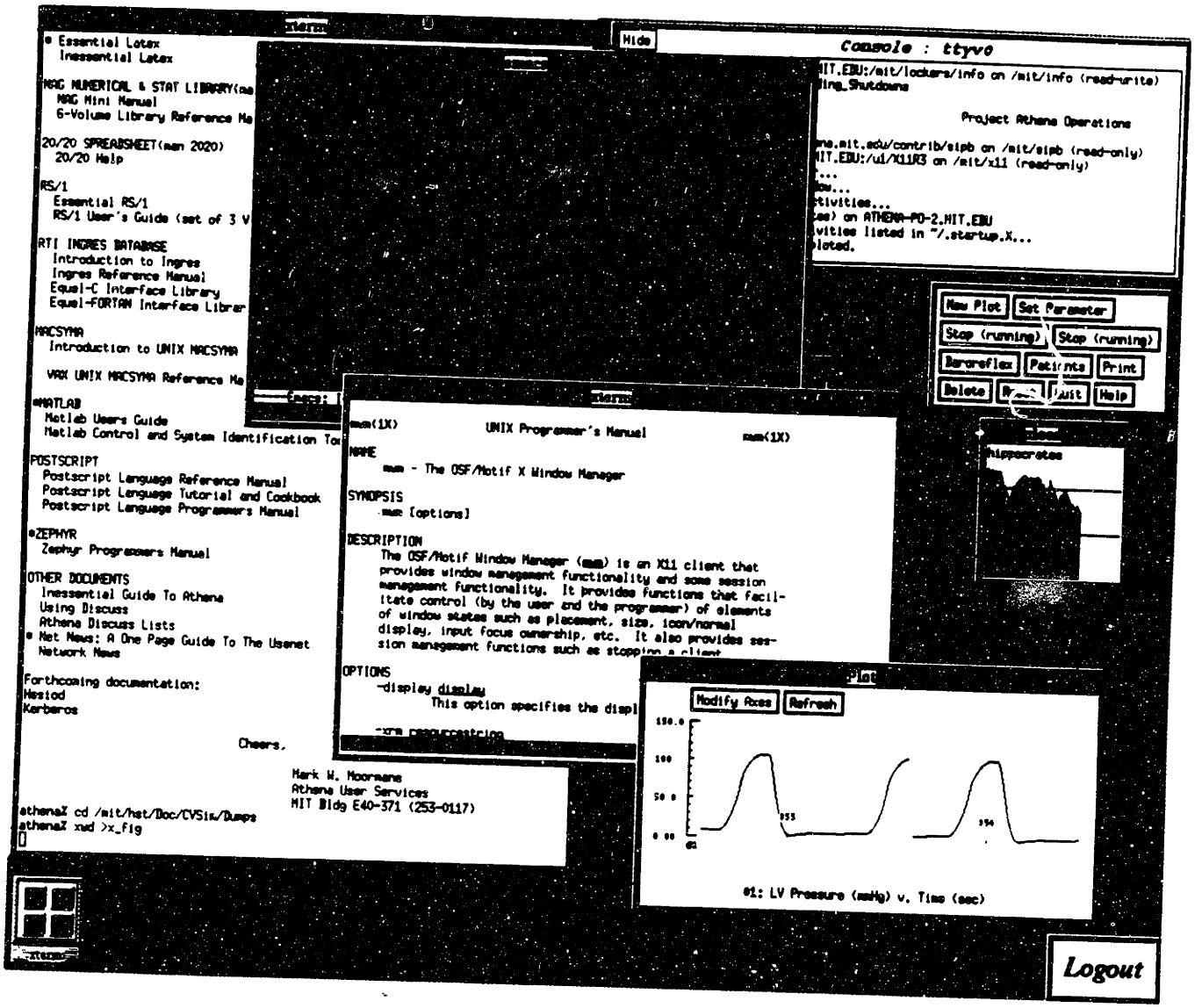


Figure 3. An example X Window System screen

The top-level windows such as **xterms** are manipulated by the window manager. The window manager allows you to move windows around, uncover obscured windows, reduce windows to icon form, etc. Project Athena has recently settled on the Motif Window Manager, **mwm** as the default.

Verify that you are using `mwm`

If each of your windows (except logout) has a small title bar with symbols in the corners, as in Figure 3, you are using **mwm**. If not, you may wish to change to **mwm** so that the course staff can help you with your window management problems. Ask a consultant (by typing **olc**, see On-Line Help below) if you have a question about this when the course staff are unavailable.

`mwm` usage summary

The two buttons at the right end of the title bar affect the size of your window. Clicking your left mouse button on the left of these two iconifies your window. Clicking the left button on the right one "maximizes" your window to fill the screen. To expand an icon, double click on it with the left mouse button, or click once and then choose the "Normal Size" entry on the pop-up window. Clicking and holding any mouse button on the box at the left end of the title bar produces a menu offering functions listed in Table 3. You can also bring up this menu by pressing either the middle or the right mouse button anywhere along the border of your window, including the title bar. Look in *Getting Started on Athena* for more information.

Function	*Region*	*Mouse Button*	*Mouse Action*
Move	Title Bar	LEFT	Drag
Resize	Edge or Corner	LEFT	Drag
Iconify	Right Corner small box	LEFT	Click
Deiconify	Icon	LEFT	Double Click
Shuffle Up	Edge or Corner	LEFT	Click
Shuffle Down	(In menu)		
Default Menu	Edge or Corner	RIGHT	Click
Menu Ops.	Background	any	Drag

Table 3. Using **mwm** to manipulate windows

Connecting to course software

There are many software packages on Athena which are accessed across a high-speed network from a central location, rather than being loaded on each workstation. Each software package is in a *locker*, a remote file directory which can be attached to your workstation's file system. Your files are in a personal locker which is automatically attached when you log in.

To attach a specific software locker, you can type **attach <lockername>** where **<lockername>** is a user name or software package locker name. A preferable way is to use the **add** command, which performs an **attach** and also allows you to use the name of the software directly as a command. For example, to use the cardiovascular simulator, type the following:

```
athena% add hst
athena% cvsim
```

Know your printer

Most Athena clusters have one or more printers. Each one has a name, which should be on the front of the printer. The printer in E25-131 is named **imhotep**, after an ancient Egyptian physician. The cardiovascular simulator allows you to make printed copies of plots on the screen. The cardiovascular simulator may ask you for a printer name if you have not already specified one, so that your plots will appear at the printer in your cluster. If the printer in your particular cluster is broken or overly busy, you may choose any other printer on the system; but of course, you must then go to another building to pick up your printouts. Feel free to refill the paper cassettes on these printers with the paper provided, but DO NOT use overhead acetates or any other foreign paper material in the printers.

Making printouts is relatively easy. To make a copy of a screen window while not in the cardiovascular simulator, type at an `athena%` prompt **xdpr -Pimhotep** (where **imhotep** is the printer name) and then click the left mouse button on the window of interest. There are several other printing facilities listed in *Getting Started on Athena.*

On-line help with Athena

Whenever you have a question or problem with Project Athena's hardware or software, except for the cardiovascular simulator itself, you can ask for help on-line. The Athena **olc** command will connect you with a program to interact via electronic messages with a live consultant. There is also a **help** command which provides menu-selected information specific to Project Athena. The Unix manual pages are also available. If you don't know how to do something, you can search the manual by keyword by typing **man -k <keyword>**, where **<keyword>** is something about which you need information. The full manual page about any Unix command **<command>** is available by typing **man <command>.**

INTRODUCTORY EXERCISES ON PROJECT ATHENA

1. Log on to Athena using your own account. See "How to Register for an Account" above, or see *Getting Started on Athena*. Note: The login process may take several minutes to completely initialize your workstation, so do it before taking time to get settled into your seat.

2. Create a second xterm window, iconify it, and make the original one larger. Deiconify the second, and put it behind the first. Move them so that they are both completely visible. Destroy both of them, by typing `exit` or by using the window manager. Make a new replacement xterm window by clicking the left mouse button on the background and releasing the button over the "New Window" selection in the pop-up menu.

3. After logging in to Athena, print something on the printer in your cluster. First, use `xdpr` to print one of your windows. Note that you must specify the printer name as in `xdpr -Pimhotep`. Repeat the command, this time clicking on the background to get a picture of the entire display.

4. *(For advanced or curious users.) Become familiar with the following commands:* ***ls, cat, more, mkdir, rm, rmdir, csh, lpr, lpq, lprm.*** *You may also wish to familiarize yourself with* ***emacs****, a text editor, and* ***scribe****, a text formatter.*

5. **ALL USERS**: Be sure to log out of the workstation before leaving. Do this by clicking the left mouse button on the ***Logout*** window in the lower right corner, and clicking on ***Yes***, or alternatively by typing `logout` to an xterm window. You must wait until the screen goes blank and the new Login window appears to be sure you have logged out successfully.

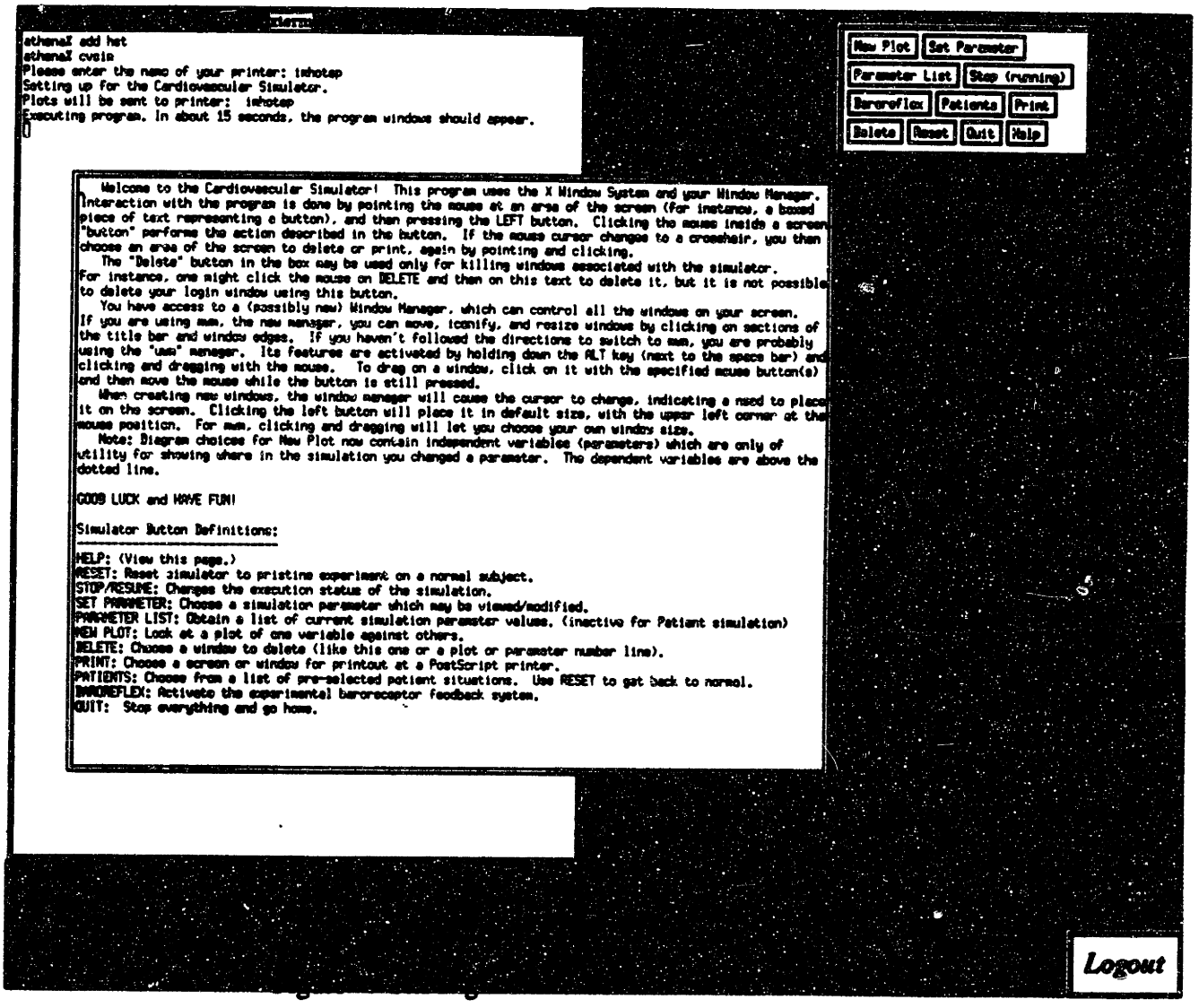
4. Using the Simulator

Starting the simulator program

At the Unix prompt (`athena%`), enter the following commands:

```
athena% add hst
athena% cvsim
```

Specify the printer name if you are asked to do so. The printer in the HST cluster is named **imhotep**. Eventually, the Cardiovascular Simulator will start up, providing a Help Window and the simulator's main command buttons (Figure 4). These rectangular outlined buttons are your main interaction with the simulator. You can use these buttons to plot data from a new catheterization, to delete unnecessary simulation windows, to alter the course of the simulation, and to print copies of your plots.



The help window

Read the text in the Help window. This will explain the use of the mouse. When you have finished reading, you may wish to iconify the window using the window manager, or delete it from the simulator command buttons. To delete it, click on the DELETE command button, wait for the mouse pointer to change shape, and then point the mouse on the help window and click again. Note: The **Kill Program** entry in the window manager menu will kill the entire cardiovascular simulator, not just the window from which it appears. You can see the Help window again whenever you wish by clicking on the HELP command button.

The NEW PLOT button and the plot menu

When you first start the simulator, it is already simulating a normal patient, but the patient has no catheters placed to measure pressure, volume, or blood flow. You may choose to plot any such variable that you wish, in effect placing a catheter in that location and hooking it up to the appropriate transducers and strip chart recorders. To do this, you choose NEW PLOT.

Setting a variable choice

First, click on NEW PLOT, which will pop up a menu allowing you to CHOOSE VARIABLE for each desired axis. Up to three variables may be plotted on the Y axis against any single variable on the X axis. The default variable for the X axis is TIME, and does not need to be changed for most plots. Locate the CHOOSE VARIABLE button for "Y axis 1" and click on it. Upon clicking CHOOSE VARIABLE, a window will pop up displaying the schematic diagram for the model. When you click on one of the anatomical areas, the electrical circuit equivalents for that area will be displayed. Click one of these for a menu of values (pressure, volume, flow) which can be measured in that location. Independent variables (resistances, capacitances, etc.) may also be plotted in order to keep track of variable changes for transient experiments. For instance, you may wish to view systemic arterial pressure versus time. Systemic arterial pressure is found in the capacitor constituent of the systemic arterial compartment. After clicking on the artery box and on the capacitor, click on the small box beside the variable name to choose that specific variable.

Choosing plot style and scaling

When you return to the axis definition table you can change the axis title, limits, and/or auto-scaling. To set a nonstandard axis limit or title, place the mouse pointer inside the small window containing the current limit or title, and simply type in the new one. Alternatively, you may select the autoscaling feature. Autoscaling reconfigures the axes to keep the entire plot within the limits of the graph. Note that if you are looking at two variables on the same Y scaling, any autoscaling may cause the axis scales to become unequal, resulting in an unaligned plot.

If you wish to place TIME on the X axis, you have the choice of strip-chart style plotting, in which the paper moves and the pen stays at the right end, or oscilloscope style, in which the pen wraps around, erasing the preceding data. The oscilloscope style is more efficient. You may also select paper speed, in centimeters per second, as the X-axis scaling option. Slow plots such as 0.1 cm/sec can be useful for tracking changes over many seconds.

Creating the plot window

Once you have finished choosing variables for the X and Y axes, click on DO PLOT from the bottom of the menu. The mouse pointer will change and a window outline

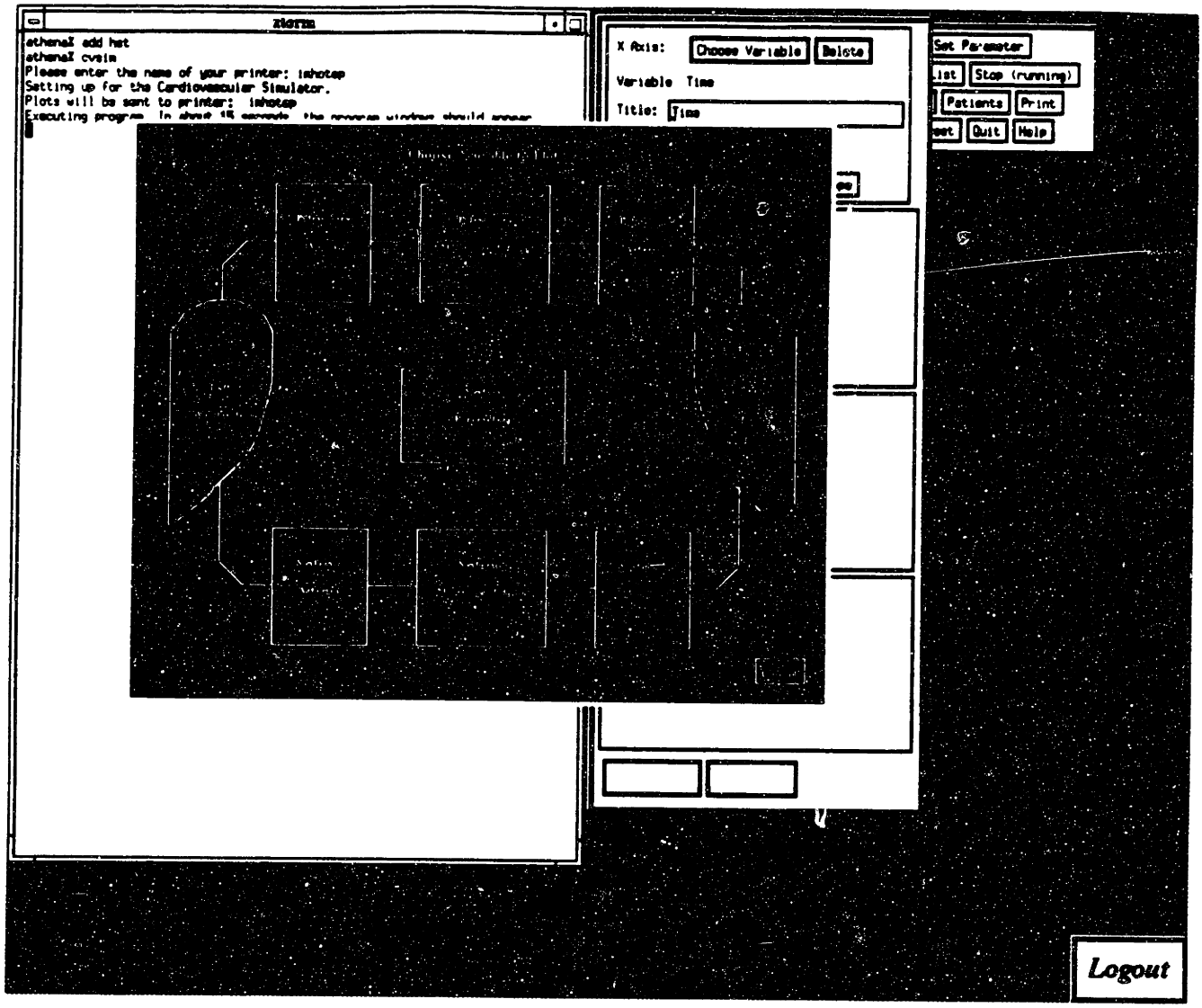


Figure 5. Choosing a variable to plot.

will appear, indicating the need to place the plot with the mouse. Do so by clicking the left button. You can change the initial size by dragging the mouse before releasing the button. The plot can be re-sized using the window manager, and axes can be modified or old data removed using the buttons provided within each plot. You may have as many plots active simultaneously as desired; however, as the number of plots increases, simulation time will begin to slow and user interaction times can become protracted.

Changing your mind

Any time the Axis Definition (NEW PLOT or MODIFY AXES) menu is visible, you may alter the information in any of the fields containing a carat ("$_{A}$") cursor. To do so, carefully move the mouse pointer into that field, and edit the field from the keyboard. Each of those small areas is a miniature text editor. You can erase the information originally in the field by moving the carat to the right (by clicking the mouse or using the arrow keys), and deleting the unwanted text with the backspace key. The first keypress will also clear out old characters. Any time a CANCEL option appears, clicking it will remove the entire menu without changing anything.

AN EXAMPLE: LV PRESSURE V. TIME

1. CLICK (using the left mouse button) while the mouse pointer is on the words NEW PLOT.
2. CLICK on CHOOSE VARIABLE under Y Axis 1. The circuit diagram pops up. The X axis is Time by default; no need to change it.
3. CLICK on the Left Ventricle box to reveal the circuit equivalents.
4. Move the mouse pointer to the capacitor, and CLICK on it when it is highlighted.
5. CLICK on the black box beside "LV Pressure." The schematic disappears.
6. If desired, you may choose a variable (e.g. Arterial Pressure) as Y axis 2 using the above techniques.
7. CLICK on DO PLOT, from the bottom of the plot creation form. That form disappears.
8. Place the mouse at the upper-left corner of an empty area of screen and CLICK to place your new plot on the screen.

Changing parameter settings

Initially, parameters of the simulation mimic those of a normal individual. These may be changed at any time with the SET PARAMETER command button. You can plot a variety of variables continuously while changing parameters such as resistance, capacitance, blood volume, and heart rate.

Creating a number line

Parameters are modified by clicking the left mouse button in a number line representation of the possible values for a variable. To create a number line, click on SET PARAMETER, then choose a parameter from the circuit diagram much as you would to select a variable for plotting.

Transient behavior vs. steady state

If the simulation is running and you have a plot showing, you can watch pressures, volumes, and flows change instantaneously as you click on different values of the parameter. The simulation will take several seconds to reach a steady state after such a parameter change. You can stop the simulation before changing parameters, then restart it after changing them with the STEADY STATE restart option if you wish to compare steady state situations side-by-side. More commonly, you will be interested in the transient behavior, which can be done on-the-fly or by stopping and re-starting the simulation using the TRANSIENT restart option.

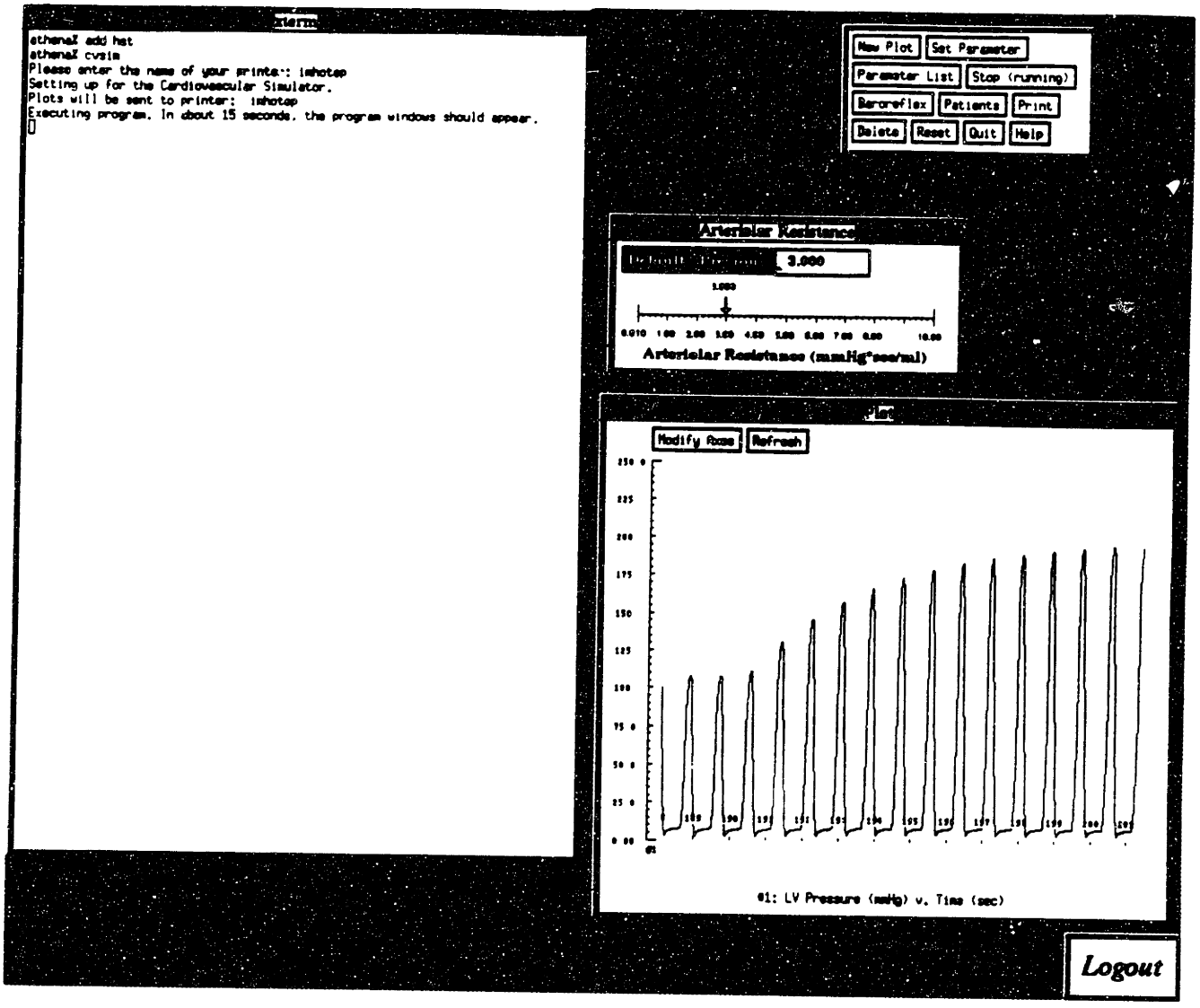


Figure 6. Changing arteriolar resistance while observing LV pressure.

AN EXAMPLE: EFFECT OF ALTERING PERIPHERAL RESISTANCE

Starting with a running plot of LV Pressure as in the preceding example, one can monitor the effects of varying peripheral resistance (Figure 6):

1. Click on SET PARAMETER.
2. The circuit diagram appears. Click on Systemic Microcirculation.
3. Click on the resistor while it is highlighted.
4. Click on the "Arteriolar Resistance" box from the menu.
5. Place the new window on an empty screen area.
6. A number-line window labeled "Arteriolar Resistance" appears, allowing you to slide a carat-shaped mouse pointer over the scale, and then to select any specific value by clicking the left mouse button. You should be able to visualize the effect on your active plots immediately.

INTRODUCTORY EXERCISES ON HEMODYNAMICS

At your initial session with the cardiovascular simulator, plot the following graphs:

1. Plot left ventricular pressure and left atrial (pulmonary venous) pressure versus time, as you change blood volume, venous capacitance, and venous zero-pressure volume. How are these parameters alike? How are they different?

2. Plot left ventricular pressure and arterial pressure versus time, while increasing left ventricular outflow resistance. What disease does this mimic?

3. Left ventricular pressure P_{lv} versus left ventricular volume V_{lv}, and P_{rv} v. V_{rv}, the classic P-V loops. Put volume on the X Axis. Try varying heart rate, then LV contractility. What properties can you deduce?

Stopping the action

The STOP / RESTART button

Using the command buttons, you may STOP the simulation and then RESUME it, either immediately or after a stable situation is detected (minimal beat-to-beat variation). You may also change multiple parameters at one point in time. You may wish to DELETE windows which you have created in the simulator. The HELP window and *Software Reference Manual* provide information on all the command buttons.

Printing

The PRINT button is generally equivalent to the **xdpr** command from Unix. In order to get a hard copy of a plot on the screen, do the following:

1. Click on the PRINT button.
2. When the mouse pointer changes to a cross-hair ("+"), select a plot, or the screen background to print the entire screen, and click with the left mouse button.
3. The workstation will beep at the beginning and end of dumping the screen, and will become sluggish until the plot has been sent to the printer. Large plots will take several minutes between the time the header page and the plot itself are printed.

Saving plots

NOT YET IMPLEMENTED.

Saving the simulation state

NOT YET IMPLEMENTED.

Using the baroreceptor control system

As described in Section 2, the simulator operates without any feedback control. All homeostatic behavior observed is the result of the stabilizing effects of compliant vessels. Physiologically, however, the cardiovascular system is controlled by a feedback regulatory system which maintains blood pressure to perfuse the body under extreme conditions. The arterial baroreflex is the most important and well-understood of the short-term homeostatic mechanisms.

When the simulator's control system is active, the parameter values for heart rate, minimum systolic capacitance $C_{l,r\,sys}$, venous zero-pressure volume V_{v0}, and arteriolar resistance R_a are used as the baseline values, to which control deviations are dynamically added during the simulation, as in Eq. (6) of Section 2. One may display these parameters, which have now become variables, in a NEW PLOT, to observe the time response of baroreceptor control. If one increases the gain values somewhat and perturbs arterial pressure, oscillatory behavior may be observed. This corresponds to an unstable system, which is limited in amplitude by a nonlinearity in the model: a maximum limit on the control signal.

Turning baroreceptor feedback on and off

The BAROREFLEX button activates the user interaction windows for the baroreflex. The baroreflex system is initially off, because it is still under development and is not a part of the current patient case study assignments. After understanding the basic hemodynamics you should experiment with the system by clicking on the BAROREFLEX ON/OFF button in the baroreceptor interaction window. Click the button again to turn the baroreflex off.

Changing control system parameters

The setpoint (desired) arterial pressure is modifiable, as are the gain and basal tone values for alpha sympathetic, beta sympathetic, and parasympathetic efferent autonomic nerves. The system maintains an error in controlled arterial pressure from setpoint which is proportional to the deviation of the uncontrolled system from normal — this is a characteristic of proportional controllers, not a bug. Click on the MODIFY REFLEX PARAMETERS button to change control parameters. The gain parameters are the constants (γ in Eq. (6)) by which the control signal (arterial pressure) is multiplied to obtain offsets for the controlled parameters, such as heart rate. Tone parameters (τ) are included, to allow an effective constant offset in the affected controlled parameters, just as if one changed the parameters directly.

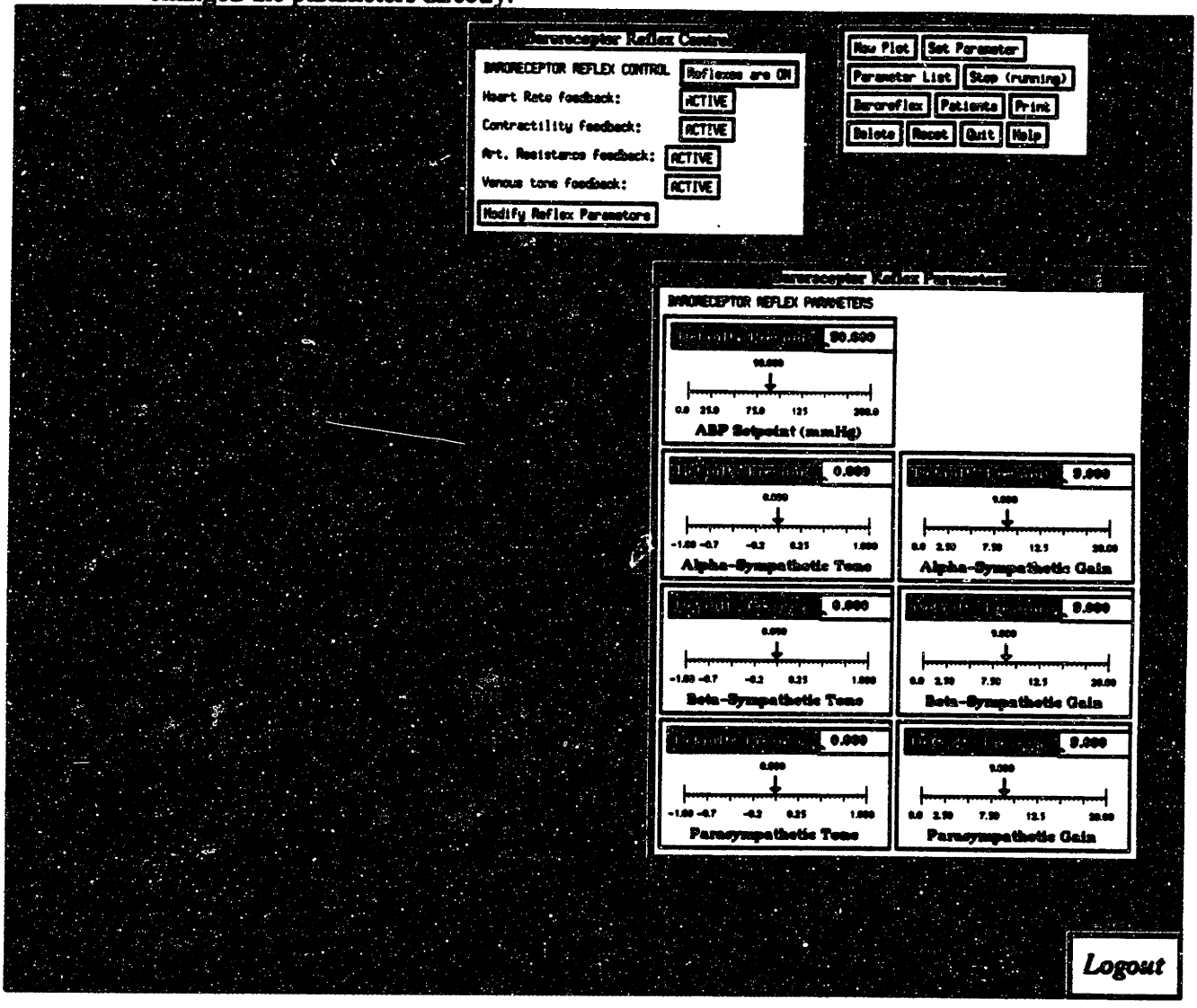


Figure 7. Baroreflex control windows

EXERCISE ON THE CONTROL SYSTEM

1. Turn on the control system with standard parameters, making sure all four limbs of the control system (HR, $C_{l,r\,sys}$, R_a, V_{v0}) are active. Start a plot of arterial pressure versus time, at 0.2 cm/sec paper rate. Cause the patient to instantly loose around one liter of blood (set parameter: blood volume), and watch the response over about 30 seconds, noting the final average arterial pressure. Compare to a similar simulation without an active baroreceptor control system.

Patient case studies

A set of clinical studies is available for you to test your knowledge of the cardiovascular system's behavior under abnormal conditions. In each simulation, the system parameters have been modified in order to simulate a specific disease. By clicking on PATIENTS you will be offered a menu of clinical cases. Each provides a brief patient history. Clicking on a case number will provide a case history and initialize the parameters to a specific pathologic model. Study the "patient" by recording hemodynamic variables at various points in the system. You should conduct your "cardiac catheterization" in such a manner as to enable you to make a full diagnosis. You may change model parameters (e.g. peripheral resistance) by percent changes in the initial values. The actual parameter values are hidden. The SIMULATOR PARAMETERS button in this case is disabled. Click on RESET to cancel the patient and return to a pristine state with a normal subject. You should be sure to keep the baroreflex control system OFF while investigating patient cases, because long-term compensatory responses are already built into the patient models.

Problems and bugs

Convergence failure

Occasionally when simulating a pathologic state or a preset example patient, the mathematical simulation will fail to converge on a plausible solution, or will not attain steady state. When this happens the user is notified of the situation via Project Athena's zephyr windowgrams. If a problem occurs during a simulation, plotting usually stops abruptly with a straight line shooting to an unrealistic variable value in response to a recent parameter change. A quick readjustment of that parameter may allow for continuation of the simulation; however, the simulation will halt itself after several retries if convergence is not reached. When this happens, it is necessary to STOP the simulation, change the offending parameter, and then click on RESTART with the STEADY STATE option. The system is sometimes quite slow at updating the screen — do not confuse this with the problem of convergence failure, about which you will be notified.

Baroreceptor control system

The baroreceptor control system is still a new and relatively unexplored feature of the cardiovascular simulator. It can be difficult to use appropriately, especially when run on a system altered to simulate a pathology. To apply it to a pathological state, set the P_a setpoint near the average current P_a. Otherwise, the control system may pin all controlled parameters to their limits while still unable to reach the desired arterial pressure.

Too many simultaneous plots

If the response to your keyboard or mouse inputs is very slow, it could be that you are trying to plot too much at one time. Try clicking on the STOP button, then wait until all action ceases. You should then be able to take corrective action (e.g. DELETE unneeded plots) before restarting the simulation.

Printer problems

You can reload printer paper yourself, using only the provided printer paper. The printers are sometimes slow to print graphics; waiting five minutes is not uncommon. Check *Getting Started on Athena* or the **man** command for information on lpq, lprm and other printer commands which allow manipulation of the printer queue.

Network difficulties

Occasionally, the network on Project Athena becomes overloaded or incapacitated. If while logging in you get the message, "Warning! Home directory is unreachable," you

should log out and try again. Similarly, you may have trouble connecting to course software or printers. This is usually a temporary situation. If the condition persists, try again a few minutes later or call the hardware hotline (3-1410).

We hope you enjoy using the simulator, and that it will give you a more intuitive understanding of the lumped-parameter model and its strengths and weaknesses. As a part of this course, you will be asked from time to time to evaluate the cardiovascular simulator based on its educational merits and ease of use. Please keep a list of problems, frustrations, and enlightening experiences while using the simulator. In addition, please report significant problems with the software immediately to your course TA. Good luck!

REFERENCES

1. Defares, J.G., J.J. Osborne, and H.H. Hara Theoretical synthesis of the cardiovascular system. Study I: The controlled system. *Acta Physiol. Pharm. Neerl. 12*(1963), 189-265.

2. DeBoer, R.W., J.M. Karemaker, and J. Strackee Hemodynamic fluctuations and baroreflex sensitivity in humans: a beat-to-beat model. *Am. J. Physiol. 253*, Heart Circ. Physiol. 22 (1987), H680-H689.

Part II: Software Reference Manual

This manual briefly describes elements of the user interface to the cardiovascular simulator. For a description of the model and examples of usage, see Part I, the *Cardiovascular Simulator User Manual.*

1. The Command Buttons

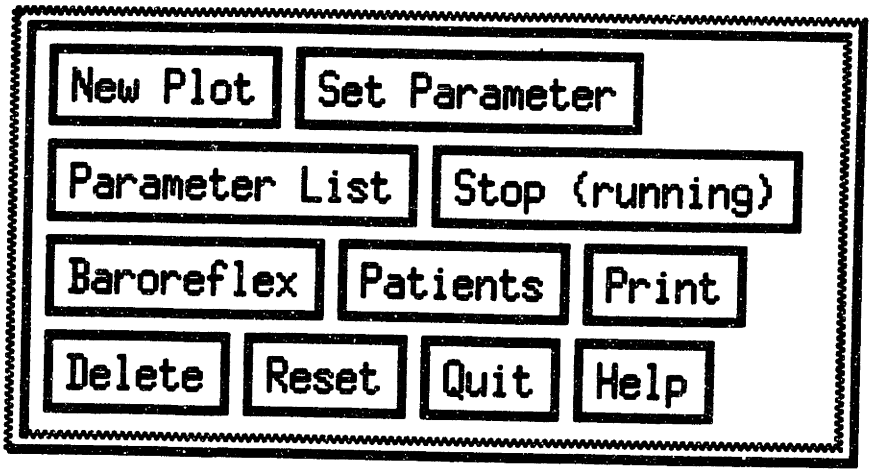


Figure 8. The command buttons

New Plot

Brings up the axis definition window, which is described below. After finishing with axis definitions, clicking on DO PLOT produces a new, independent plot.

Set Parameter

Brings up the parameter choice menu, described below. Only independent variables are shown. Choosing a parameter produces a number line which may be used at any time to change the value of that parameter.

Parameter List

Displays a complete list of basic simulation parameters, excluding the control system. When a parameter has been changed from the default, it is marked with a "+" or "-" and the new value.

Stop Simulation

Halts the mathematical simulation engine, stopping any new plotting. Useful for freezing the current transient behavior on the screen and for changing multiple parameters at one point in time.

Start Simulation

Restarts the mathematical simulation engine after it has been stopped. The user is asked whether to resume immediately in continuity with the preceding simulation (TRANSIENT) or to establish steady-state behavior (STEADY STATE), in which case the simulator waits until each successive beat is nearly the same before resuming the plotting of data.

Baroreflex

Brings up the baroreflex control window. A main REFLEXES ARE ON/OFF button will turn reflexes on and off. There are four limbs to the reflex (heart rate, contractility, peripheral resistance, and venous zero-pressure volume), any of which can be made active or inactive by clicking on the corresponding ACTIVE/INACTIVE button. To modify reflex parameters, click on the BERORECEPTOR REFLEX PARAMETERS button. This brings up a set of number line parameter modification number lines which may be used to change reflex parameters in the same manner as the other parameters are modified.

Patients

Brings up a list of numbered patient cases. Clicking on one of the cases sets up a predetermined patient simulation, and displays a clinical case history. It is the responsibility of the student to make the appropriate investigations to determine the diagnosis. The parameter list is disabled, and parameter modification is altered to reflect only percentage changes from the patient's original parameter values.

Save Plot

NOT YET IMPLEMENTED

Save State

NOT YET IMPLEMENTED

Print

The mouse pointer changes to a bullseye, at which time the user must point at a window and click the left mouse button. Clicking on the background produces a printout of the entire screen. All printouts are made on the printer named at the beginning of the session.

Delete

The mouse pointer changes to a cross-hair, at which time the user must point at a simulation window and click the left mouse button. This only works for the following simulation windows: plots, number lines, parameter lists, the baroreflex control windows, and the help window. The Kill Program menu selection, in contrast, will kill the entire simulator.

Reset

Resets the simulator state entirely, returning all parameters to those of a normal patient. After clicking, the user is asked to verify the action or to click CANCEL.

Quit

Exit the program. A confirmation is requested; CANCEL returns the user to the simulator. All work not printed or saved will be lost. The user remains logged into the workstation, and must log out before leaving.

Help

Produces a window containing brief help information about the cardiovascular simulator.

2. The Plot Axis Menu

The plot axis menu appears when the user requests a new plot or to modify an existing plot. This menu allows the user to choose variables for one X and three Y axes to be plotted on the same graph.

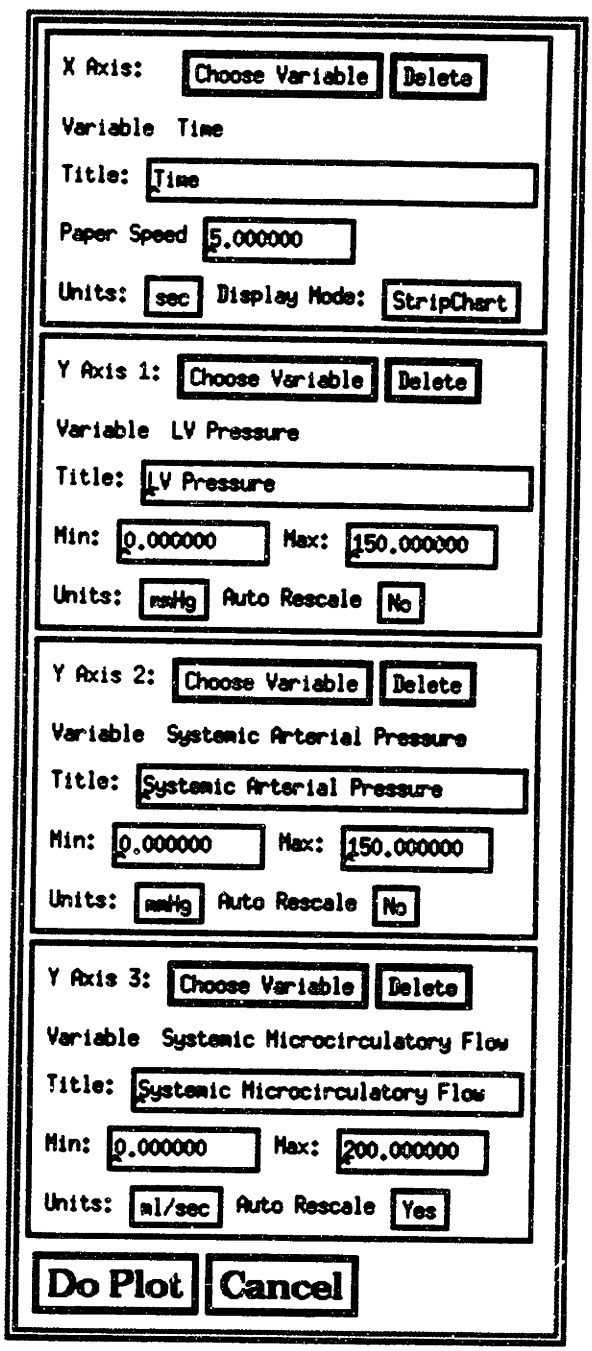


Figure 9. The Plot Axis Menu

CHOOSE VARIABLE button

This button brings up the circuit diagram parameter choice menu (see Section 3) from which a simulation variable can be chosen. After choosing the variable, default values for plotting limits and the axis title are filled in.

Minimum and maximum values

These are filled in after choosing a variable, but may be changed. To set your own limit, place the mouse pointer inside the small numeric window and type in the new value, using the arrow keys and backspace to move over and delete existing text.

Axis label

The axis label can be modified similarly to the minimum and maximum values.

Auto-scale

The auto-scale button turns auto-scaling on or off. While on, any time data exceeds plot boundaries, the axis will be recomputed to show all data. Note that the plot will never reduce the maximum or increase the minimum values; auto-scaling only brings extremes into view.

Strip-chart vs. oscilloscope mode

When the X-axis is TIME, a button in the X-axis section allows the user to select either *strip-chart* mode or *oscilloscope* mode. Strip-chart mode produces the impression of a moving strip of paper drawn across the screen from the right edge of the window. Oscilloscope mode redraws the data continuously from left to right. Oscilloscope mode is more efficient.

Paper speed

When the X-axis is TIME, the minimum and maximum values are replaced by a paper speed window. Paper speed can be changed from default (like the minimum and maximum values), in order to see long-term changes or to carefully explore the fine structure of a waveform.

DO PLOT and CANCEL buttons

The new plot or previous plot are not updated until the NEW PLOT button is activated. CANCEL will remove the menu without making any changes to existing plots.

Changing your mind

All buttons remain active, even after an axis has been defined. One can choose a different parameter by clicking on CHOOSE PARAMETER, or remove that axis definition from the plot by clicking on the DELETE button for that axis. CANCEL will completely remove the plot axis menu and cancel all changes.

3. The Parameter Choice Window

When choosing a variable to plot on an axis or choosing a parameter to modify, the user is presented with a circuit-diagram equivalent of the cardiovascular system. To navigate through this circuit, the user clicks the left button of the mouse on blocks, components, and subcomponents of this diagram.

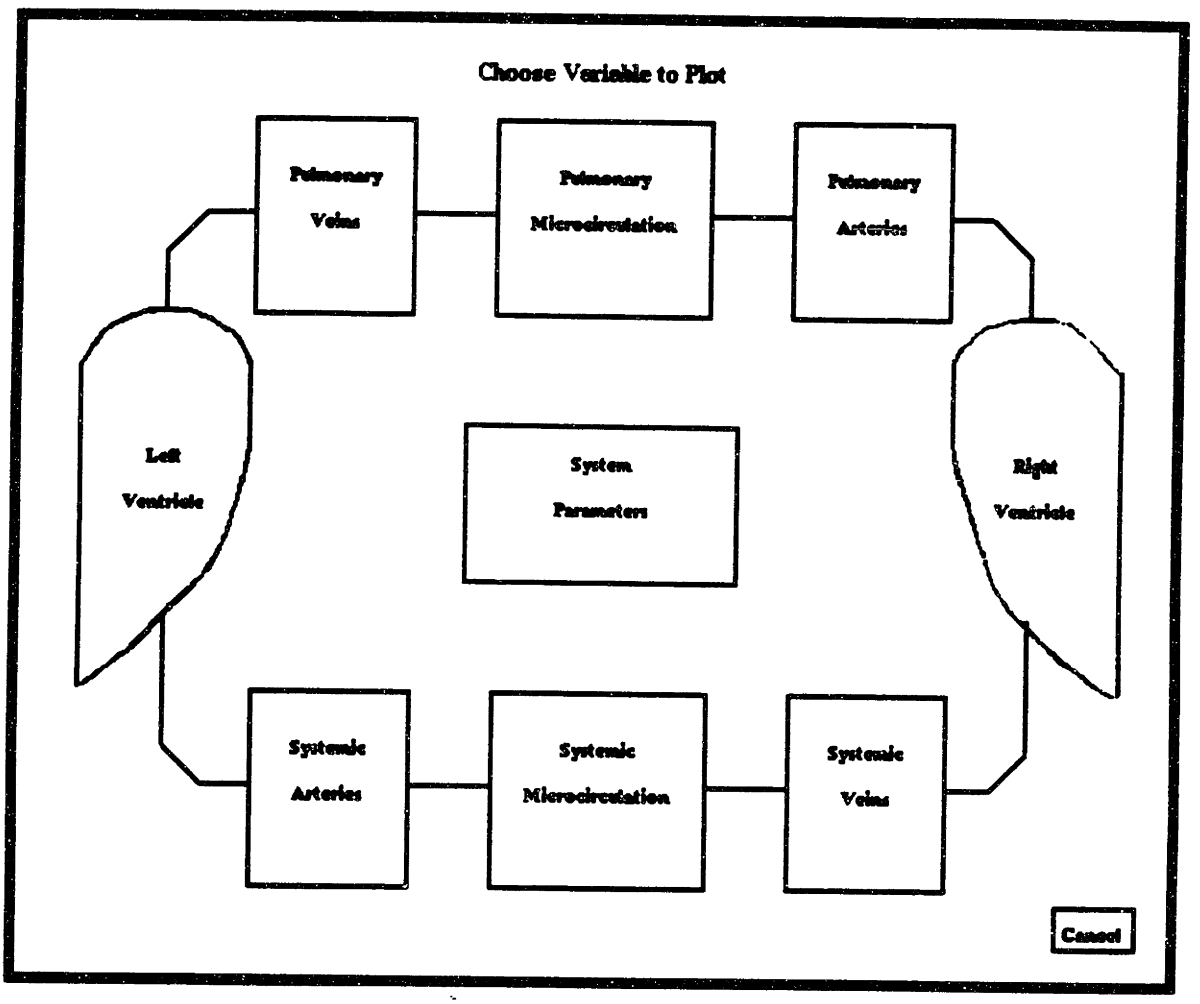


Figure 10. The Parameter Choice Window.

Choosing a functional compartment

One initially clicks on a functional body compartment, such as the left heart, systemic microcirculation, or the system parameters box. This will replace the compartment name with a diagram of the sub-circuit contained therein. Moving the mouse pointer out of the compartment will cancel this choice.

Choosing a constituent

Once a compartment is chosen, the user selects a constituent such as capacitor or resistor by pointing the mouse on it and causing it to light up. Clicking on this constituent will produce a list of variables and parameters associated with that constituent. Capacitors have associated with them pressure and volume variables, and capacitance and zero-pressure volume parameters. Resistors have an associated flow variable and resistance parameter.

Choosing a parameter

A parameter is chosen from the list by clicking the mouse while pointing at the small box to the left of the parameter name. Clicking on CONTINUE allows the user to choose a different constituent.

Changing your mind

The CANCEL button at the lower right will remove the parameter choice window with no change to the simulation. Clicking on CONTINUE in a parameter list allows the user to choose a different constituent or to CANCEL the process.

4. Implementation on Project Athena

Currently, the cardiovascular simulator is installed on the Project Athena system at M.I.T. The following paragraph is intended only for the curious or advanced user who wishes to know more.

The cardiovascular simulator is written in C, using the X11 library, Xt toolkit, Athena widget set, and two additional widgets built for the simulator: Plot and NumberLine. While at a Project Athena workstation, one can attach to the Athena NFS locker **hst**, which contains shell scripts and IBM RT and Vax executables in /mit/hst/rtbin and /mit/hst/vaxbin. Because NFS does not cache executing code locally, the programs are copied to the local machine before running, which improves performance. This is also the course software development locker, with source code in /mit/hst/src.

APPENDICES

Appendix A: Model Parameters
Appendix B: Getting Started on Athena

Appendix A: Model Parameters

Uncontrolled model

The mathematical engine for the cardiovascular simulator solves a set of six coupled linear time-varying differential equations of the form

$$dx/dt = Ax + b \tag{7}$$

by a fourth-order Runge-Kutta method. Eq. (7) can be interpreted in matrix form as Kirchoff's Current Law for each of six capacitors in the system, where **x** is a vector of six pressures (the state variables), **A** is a time-varying square matrix describing the connections between adjacent compartments, and **b** is a vector containing transthoracic pressure offsets.

The following parameters define the state of the simulation:

$C_{l\,sys}$	LV min systolic capacitance	V_{l0}	LV zero-pressure volume	R_{lo}	LV outflow resistance
$C_{l\,dias}$	LV max diastolic capacitance	V_{a0}	Arterial zero-pressure volume	R_a	Arteriolar resistance
C_a	Arterial capacitance	V_{v0}	Venous zero-pressure volume	R_v	Venous resistance
C_v	Venous capacitance	V_{r0}	RV zero-pressure volume	R_{ro}	RV outflow resistance
$C_{r\,sys}$	RV min systolic capacitance	V_{pa0}	Pulmonary arterial zero-pressure volume	R_{pa}	Pulmonary microvascular resistance
$C_{r\,dias}$	RV max diastolic capacitance	V_{pv0}	Pulmonary venous zero-pressure volume	R_{li}	LV inflow resistance
C_{pa}	Pulmonary arterial capacitance	bv	Total blood volume	P_{th}	Transthoracic pressure
C_{pv}	Pulmonary venous capacitance	hr	Heart rate		

Baroreceptor control system

The baroreceptor reflex response is computed by the following equations, which are calculated once every 500 milliseconds:

$$s'[n] = 18\arctan[(s[n] - s_0)/18] \tag{8}$$

$$I[n] = I_0 + \sum_{k=0}^{L}(a[k]\ s'[n-k]) \tag{9}$$

$$R[n] = R_0 + \frac{1}{C_a}\sum_{k=0}^{L} b[k]\ s'[n-k] \tag{10}$$

$$C_{l,r\ sys}[n] = C_{l,r\ sys\ 0} + \sum_{k=0}^{L} c[k]s'[n-k] \tag{11}$$

$$V_{v0}[n] = V_0 - \sum_{k=0}^{L} d[k]s'[n-k] \tag{12}$$

Variable Definitions:

n:	Discrete time index
s_0:	Arterial pressure setpoint
s:	Arterial pressure
s':	Adjusted arterial pressure signal before time-averaging
I_0:	Nominal inter-beat interval
I:	Controlled inter-beat interval
a:	Impulse (unit sample) response for heart rate
b:	Impulse response for peripheral resistance
c:	Impulse response for contractility
d:	Impulse response for venous blood pool
R_0:	Nominal peripheral resistance
R:	Controlled peripheral resistance
C_a:	Arterial capacitance
$C_{l,r\ sys\ 0}$:	Nominal minimum left or right ventricular systolic capacitance
$C_{l,r\ sys}$:	Controlled minimum left or right ventricular systolic capacitance
V_0:	Nominal venous zero-pressure volume
V_{v0}:	Controlled venous zero-pressure volume

Appendix B: Getting Started on Athena

See additional handout.

Appendix B

Questionnaire Responses

This appendix lists the answers to the written comments on questionnaires received from two populations of students: undergraduates in the fall of 1989 and graduate M. D. and Ph. D. candidates in the spring of 1990. Answers to the agree-disagree statements are summarized in the Utilization chapter, and are not included here.

B.1 Fall group: Undergraduates

A comprehensive list of short-answer responses for the eleven questions in the fall questionnaire and general comments follows. Questionnaire respondents are numbered from one to eight.

- What is your impression of the cardiovascular simulator?

    1. It was in general a pretty good tool and helpful in understanding. Getting familiar with the operation of the software was the initial problem though.
    2. The simulator was really helpful for me in understanding the cardiovascular system.
    3. It is one of the better X-Windows applications I have yet seen. It works well and runs pretty fast, with relatively few bugs (not reproducible). As a learning tool it is an excellent display environment. Better than watching a lecturer on a chalk board.
    4. It is a useful tool. I think it is an excellent beginning to a project that could be extremely worthwhile.
    5. I thought it was useful.

6. It can be useful for someone who is interested in using computers, who understands the basics behind using similar programs. I never had that background so it took me a long time to understand what I was looking at.
7. An excellent simulation of a complex problem that could be a little more user friendly to really stupid people.
8. Generally, I liked it and thought it was well done.

- What problems did you have using the program?

1. The program itself worked well as expected. However the Motif window system posed problems, especially with other simulators.
2. It would be helpful if we can display both the patient and the normal person's information on the same graph without having to erase the normal person's data.
3. Occasional bugs with clearly weird results. When this happened, "reset"ing the system always worked fine. The bugs were not reproducible.
4. Not being able to look at the patient and case at same time. Not being able to look at graph values exactly.
5. You couldn't change which printer you can use once you've started the program.
6. Although many parameters were plotted against time, after stopping the program I did not necessarily want it to resume steady state but take off from where we left off.
7. My main problem was knowing what to do next. It was also very time consuming to explore the system to understand it.
8. Very few and very minor: a) Not enough memory on the RT's causing a lot of disk swapping. b) Occasional crashes (very few) c) Printout scaling. I already talked to you.

- What did the simulator help you to learn about physiology?

1. Helped one to understand the interactiveness of the different parameters, and the effects of changes in the body. It was nice to see the slow changes resulting from perturbations.

2. It showed be how changes in different part of the system affect the rest of the cardiovascular system.
3. If we had had more time, exploring baroreceptors would have been very useful. As it was, the case studies were the best examples of physiologic problems. It was most useful in displaying interrelationships of variables.
4. It helped me learn a lot about my case #2, and the effects of a change in resistance on the CV system.
5. Control mechanisms.
6. Not a whole lot.
7. How resistance and pressure and flow work together.
8. Hard to say. Helped me to learn about diagnosis and where/what symptoms to look at.

- What did the simulator help you to learn about modelling?

1. It made one realize the limitations of the model, and made you identify what the model actually can show.
2. It showed me that modelling is not perfect.
3. It's a good example of a multi-variable system. It is always helpful to watch the reactions of a model to parameter changes.
4. It showed that models are [not] perfect, and the difficulties in making a good model.
5. That there are some things that you cannot simulate in a model.
6. Changing capacitance / contractility
7. That it works pretty well in most situations.
8. This is where the simulator was helpful. Showed us the complexity of modelling such a complex control system and gave us a feel for the approximations which one can make.

- What else did you learn?

1. Learned that one must be patient w/computers — they don't always listen to reason.
2. [No answer.]
3. [No answer.]
4. [No answer.]
5. What might be done to normalize a patient with an abnormal pathology.
6. Mainly I spent time figuring out how to plot the different types of curves that allowed me to measure useful parameters.
7. That [modelling] can also be misleading. For example the modelling of patient #3 involved shoving massive flow through the microcirculation when the opposite was actually the case.
8. Mostly heart stuff, from Guyton really.

- How did the simulator confuse you?

1. The existence or rather non-existence of the baroreceptor controls where confusing, since their control responses were simulated.
2. I didn't know that it takes time for the simulation to reach steady state.
3. Setting up graphs was not that straightforward, at first. Also, as beginners, our startup time was long. Perhaps we needed a stronger CV background first. It is a somewhat imposing system at first glimpse.
4. It didn't.
5. Not really.
6. The timing — I didn't like that fact that I was limited to only showing flow through certain central organs because I didn't know which parameters were supposed to be similar. Having the option of seeing these myself would've helped.
7. It was pretty straightforward, not too confusing.
8. Not, if anything it gives you a chance to work out ideas and shows misconceptions when what you expect didn't happen.

- How would you change the teaching of hemodynamics?

1. [No answer.]
2. [No answer.]
3. [No answer.]
4. Do more cases and group projects, where we all can discuss — learn from one another. The discussion on the first day of presentations were excellent. More of this.
5. It's fine.
6. I wouldn't use the simulator until I understood the basics.
7. I would work through a demo patient with the computer with the group before they are sent out on their own.
8. [No answer.]

- What simulator enhancements would be the most useful to your learning?

1. It would be nice to be able to see two patients at one time, or at least a comparison w/ a normal patient. Also modifications in the plotting procedure, i.e. identification of points, labelling — annotation would be nice.
2. [No answer.]
3. I would provide: step-by-step, computer-controlled tutorial for setting up P-V loops, etc. Also, if a single button click could pull up standard charts, auto-scaled and labelled it would be great. Have a button labeled "P-V loop" or "Pressure vs. Time" etc. Also, two patients at once would be great.
4. Be able to have multiple patients graphics on the screen at the same time. Improve graphing features. For tutorial purposes, create a demo that allows you to hit one button to make one change and to show the most important plots with explanations.
5. Being able to do more than one patient at the same time and compare outputs.
6. see question 6.
7. Demonstration tutorials to step you through a diagnosis of a sample patient. Such as a "try this next" type deal.
8. Better understanding of what control state the patients are in.

- How would you counsel another student about the CV simulator?

  1. A demonstration of the CV simulator would help one grasp how to work it better. Discovery on your own is nice but time-consuming and often frustrating.
  2. [No answer.]
  3. See tutorial, above. I think seeing is best instead of writing about what you will see.
  4. Just start off with the project and learn as you go along.
  5. Just play around with it and see what it can do.
  6. Show them how to make plots and how to make changes. An intro that explains the limitations and the purpose (in its limited use) would be useful. A sample session and how to switch between different options (the ordering).
  7. Know what you want to try before you sit down at the terminal.
  8. [No answer.]

- How was your time spent while working on the simulator projects?

  1. A lot of my time was spent just seeing "What-this-thing-does" or "what happens when you change this"
  2. [No answer.]
  3. Switching between patient and normal!
  4. Too much time spent setting up certain changes. Not enough time observing and thinking.
  5. I had to use more than one workstation in order to work on the project.
  6. I read the history, tried to think of what would change to prove it. Then I did the graphs.
  7. A lot of lost time looking at irrelevant parameters.
  8. Running things to compare to normal.

- What feelings do you have about the simulator projects?

  1. [No answer.]

2. It was well spent except at the very beginning while the print program wasn't working.
3. I liked it a lot, but I've used computers and Athena for a long time. It needs many hooks to help neophytes along.
4. Projects are excellent! With an improved simulator, we could do a lot more smaller projects, where we could learn about some specific cases w/ the simulator, but not go through all the investigations necessary to find out where to look for problems.
5. They're useful, but limited.
6. I guess I started out not knowing how to do simple things so I didn't like using it. It was time consuming (something that could've been helped by a tutorial).
7. [sic] There a good experience.
8. Overall good, a bit of a time sink to do well. I think the course should use Guyton or something like that instead of the books we use. It doesn't cost much more and is a more useful book in the future, especially for premeds.

- General comments:

1. Recommendations for the CV Simulator: Be able to plot 2 cases on same axes. Be able to pick out point values from graphs. Allow printer to be changed from within program. Be able to create open circuits and short circuits at nodes in the model. Make "key" window movable. Be able to cancel "PRINT". Label and put scale on time axis. Insights gained: Gained a lot of insight into the modelling aspect of the simulator, and gained a good deal of insight into the cardiovascular system. I think the simulator is a very worthwhile endeavor, and this lab was an excellent experience, especially with the discussion that followed at the presentations. I would hope that the simulator continues to improve and becomes an even larger part of the course in the future.
2. On the Cardiovascular Simulator: The patient case studies were an excellent way to demonstrate the uses of the simulator. However, instead of attempting to jump right in with your own patient it would have been extremely beneficial to be able to work through a sample patient with some sort of guidance. This could be an interaction

tutorial or having the instructor work through one of the patients first for the class using the simulator. This would help a lot. I also think a list of parameters on the screen would be helpful to go along with the picture of the model where you click on a particular part to see the parameters.

B.2 Spring group: Ph. D. and M. D. candidates

A comprehensive list of short-answer responses for the four questions in the spring questionnaire follows. Respondents are numbered from one to 25.

- How did the cardiovascular simulator impact your learning of hemodynamics?

1. It reinforces the usefulness of the PV loop.
3. Not much.
4. Helped clearly understand effects of each parameter.
6. Helped me see trends, but I wish I could have understood why things were happening more on a physiological basis.
7. Helped.
8. Very worthwhile for understanding individual peripheral and cardiac effects, especially as part of a complex combination of CV changes.
9. It was helpful in learning the hemodynamics. An excellent program!
10. I think it was very useful. It was good to see how all the parts of the model fit together. It was particularly useful to compare my prediction with what the computer actually did.
11. It was a fun way to see effects of paramters on hemodynamics
13. I learned the majority of what I know about hemodynamics from the simulator rather than from the readings.
14. Have parameters monitored all at once; be able to watch transient adjustment to steady state.

15. Helped to see pressure v. time plots and PV loops to understand relationships between LV, aorta, etc.
16. The simulator gave me a better understanding of cardiovascular hemodynamics response to intervention or changes of system properties.
17. Simplifies thru model, gives a framework for understanding the physiology.
18. PV loops were especially instructive and easy to manipulate on the simulator.
19. The CV simulator has helped my studies of hemodynamics very much.
21. It was very good at integrating many variables and teaching their relationships to each other. A dynamic model is certainly needed to really comprehend a dynamic system.
22. Helped.
23. It didn't have much of an effect on my understanding of hemodynamics. It was interesting for self-examination, but didn't help me evaluate functionality.
25. I think that I had a good background in hemodynamics before I started the class, so that the simulator had a marginal impact.

- Evaluate the way in which the simulator was incorporated into the teaching of HST–090.

1. It was fairly well done, except that qualitative problem should be used instead of quantitative — it was a waste of time trying to get area under curves.
3. Usually ok to model similar simulations but more complex problems lead to simulations which are difficult to understand.
4. Cases paces into memory very useful.
6. I think it is helpful but a better explanation of why parameters change should supplement the simulator.
7. Use for problem sets and in coordination with case studies was appropriate and illuminating.
8. Good.
10. It was done well. We were made to use it early, for one of the problem sets, and I think this early introduction made me disposed to using it throughout the course.

11. O.K.
13. Very good! I was a bit intimidated about using this simulator at first. The intro. session with other students was really helpful, and I liked how the first homework assignment told us exactly how to set the parameters.
14. Assignments seemed a bit simple but very useful with exam question and other case studies.
15. Good.
16. It is good.
17. Should have been used more earlier, before mid-term.
18. Complemented lecture material ad hemodynamic issues extremely well.
19. Very helpful. Makes the studies much more enjoyable.
21. Very well. One problem per homework set was appropriate.
22. Pretty useful.
23. The simulator was very easy to use and clearly put together.
24. Effective.
25. It should not be compulsory, as it were.

- What were the biggest headaches in using the cardiovascular simulator?

1. None. It was pretty user friendly.
3. checking in/out, crashes, and holding screens.
4. Time. Too slow when many people using, especially printouts.
5. The printer.
6. Sometimes it seemed slow.
7. Occasional crashes, e.g. when accessing patient data.
8. Inability to see parameters in preset patient simulations. Glitches (random lines) in graphics that sometimes made it difficult to see transients.
9. Waiting for printouts.

10. I didn't find any major problems. Sometimes fitting all the little windows onto the screen wasn't very convenient.
11. Cases were not alterable.
13. My biggest headaches took place when I was adjusting several parameters at the same time. I did this for the CV take-home midterm. It was hard for me to keep track of just what exactly I had just changed. Then, after I changed some more parameters, I couldn't get back to a previous PV-loop that I had done.
14. Getting over to M.I.T. [from Harvard Medical School]
15. The baroreceptor mechanism I found not helpful. Overall, though, its a well-designed and easy to use program.
17. None really.
18. Not many. Sometimes the wait time for printing was a little long.
19. The program runs slowly on the system.
22. The control (baroreceptor) part was not as useful as the rest.
23. Generally, there were no big problems. No bugs and the program didn't crash.
24. None.

- How could we do it better?

1. Help in area estimation.
4. Provide list of altered parameters in cases.
6. Maybe give a case or an example in the computer which could better incorporate the different objectives of learning the physiology.
9. Maybe you could incorporate more "fixed" cases, that is, all the student would have to do is choose dilated cardiomyopathy or aortic stenosis and the simulator would do the rest, showing acute and chronic hemodynamic responses along with a written explanation of what happened.
10. I know this is a cop out, but I can't think of any significant changes that would be worth doing.

11. Ability to see cardiac function / venous return curves. Also, on/off baroreceptor reflex function needed.

13. Maybe you could put in an option to save certain sets of parameters. That way, with a click of one button, the student could immediately work on something he/she had been working on before without having to reset all of the parameters for everything! Also, I wish that when restarting a stopped plot (either transient or just steady-state) it would pop up faster! I guess this is a limitation of Project Athena.

14. Instead of "plug in these numbers, print, and hand-in," home sets could be more exploratory and open-ended.

16. It would be better for purpose of student learning if cvsim also provides functions of explanation of hemodynamics, changes effects on system, clinical significance, etc.

17. It was good for the basics.

18. It would be nice to incorporate cardiac output and venous return curves into the simulator.

21. It would be helpful to be able to manipulate variables in the case studies to evaluate the effects of treatments. Also, a signal that would remain on the screen to indicate whether baroreceptors are "on" or "off" would be nice.

22. Add an inductor to the circuit?

23. Include some on-line documentation explaining effects we are seeing, ex. something like the info in the ECG program.

25. Port it over to a Mac with more features.

- General comments:

2. Early in the course when the simulation is most used, the basic concepts that the simulation is trying to represent are unfamiliar. Perhaps we should explore the real system before we tinker with an unfamiliar toy.

7. A graph paper grid option for plots.

9. I'm still not sure when or how the baroreceptor option would be used — maybe a problem on this would help.

13. Overall, this simulation is really a useful learning tool and more importantly, user friendly!

15. Simulator was fun to use. Need to find a way to do cardiac and venous return curves.

23. Overall it was a good program — I just personally learned hemodynamics better from text. In other subjects, e.g. understanding ECG, I got less from reading and more from computer work.

Appendix C

CVSIM Program Listing

This is a program listing of the files necessary to build the cardiovascular simulator for X Version 11, Release 4. If compiling for Release 3, define "X11R3" to the compiler to invoke the appropriate backwards-compatible constructions; some additional changes may also be necessary. There are numerous other compile-time options switchable by defining symbols; for example it is possible to substitute fixed-time Runge-Kutta integration for the default variable-timestep (`rkqc.c`) integration.

The listing is divided into modules, which have minimal and well-documented connections to one another. The two X Toolkit widget modules, NumberLine and Plot, obey the `Xt` program structure conventions, with a public-exporting header (ex. `Plot.h`), a private header (ex. `PlotP.h`) for subclassing, and standard widget interaction rules including communication by the resource manager.

C.1 Main program

Main.c contains the main() function which sets up the X Window System, reads the default X resource file in cvLib/XResources , and begins the simulation. Interface.c defines the interaction between interface objects, including the button box which generates the instructions to open new windows. Help information is taken from cvLib/help.txt and displayed.

Interface.h

```
extern Widget InterfaceCreate();

extern Widget DialogCreate(/* Widget parent; char *desc, val */);
extern void PlaceDialog(/*Widget w, parent */);
extern void KillDialog(/* Widget w; caddr_t client, call */);
```

copyright.h

```
/*
 *$Source: /mit/hst/src/main/RCS/copyright.h,v $
 *$Header: /mit/hst/src/main/RCS/copyright.h,v 2.2 90/08/18 19:14:56 tldavis Exp $
 *
 * Copyright 1990 Timothy L. Davis
 * All rights reserved.
 *
 */
static char *copyright_notice =
    "Copyright 1990 Timothy L. Davis.  All rights reserved.";
```

Interface.c

```
#ifndef lint
static char *rcsid="$Header: /mit/hst/src/main/RCS/interface.c,v 2.6 90/08/28 17:05:52 tldavis Exp $";
#endif lint
/*
 * interface.c:  Copyright 1990, Timothy L. Davis
 *
 * $Source: /mit/hst/src/main/RCS/interface.c,v $
 * $Log:    interface.c,v $
 * Revision 2.6  90/08/28  17:05:52  tldavis
 * minor bugs
 *
 * Revision 2.5  90/08/28  15:57:09  tldavis
 * Removed need for IntrinsicP.h (to help lint).
 *
 * Revision 2.4  90/08/28  14:51:27  tldavis
 * linted.
 *
 * Revision 2.3  90/08/18  18:45:53  tldavis
 * Final sundry interface bugs, fixed up for resources file usage.
 *
 */
#include <X11/Intrinsic.h>
#include <X11/DialogDefs.h>
#include <X11/Shell.h>
#include <X11/Xaw/Box.h>
#include <X11/Xaw/Command.h>
#include <X11/Xaw/Form.h>
#include <X11/Xaw/AsciiText.h>
#include <X11/Xaw/Dialog.h>
#include <X11/Composite.h>
#include <X11/cursorfont.h>
#include "../sim/CVDefs.h"
#include "../plot/Plot.h"
#include "../form/PlotForm.h"
#include "../al/NumberLine.h"
#include "../circuit/Circuit.h"
#include "../data/Parameter.h"
#include "../event/Event.h"
#include <stdio.h>
#include <strings.h>
#include <signal.h>
#include <fcntl.h>

extern double  getval(); /* from eqns.c */
extern SIMULATION *presentsim(); /* from eqns.c */
extern char *CVLib; /* Main.c defines location of files */
extern void ReflexesOff(); /* from ../sim/reflex.c */
extern Boolean reflexesOn;
extern char *getenv();

extern Display *dpy;
extern Window root;
extern int screen;


/* Private Definitions. */

int patientActive = 0;
typedef struct {
    Widget buttonBox;
    Widget buttons[21];
    Widget patients;
    int simulating;
    int patientActive;
    int administration;
} CVDataRec, *CVData;

/*FORWARD*/
static void Help(), SimToggleOnOff(), SimParams(), NewPlot(), DeletePlot(),
    PrintPlot(), Quit(), Patients(), ResetButtons();
void ReflexButtons(); /* from ../sim/reflex.c */
void ShowParameters();

static char *Buttons[] = {
  "New Plot", "Set Parameter", "Parameter List", "Stop (running)",
  "Baroreflex", "Patients", "Print", "Delete",
  "Reset", "Quit", "Help", NULL,
};
/* We refer to some buttons by index number below, so here they are: */
#define LISTBUTTON 2
#define STOPBUTTON 3
#define REFLEXBUTTON 4

static void (*ButtonProc[])() = {
  NewPlot, SimParams, ShowParameters, SimToggleOnOff,
  ReflexButtons, Patients, PrintPlot, DeletePlot,
  ResetButtons, Quit, Help,
};

/*PUBLIC*/
```

```
Widget DialogCreate(parent, desc, val)
    Widget parent;
    char *desc;
    char *val;
{
 Widget shell, dialog;
 Arg a[5];
 int i = 0;

 shell = XtCreatePopupShell("Popup", overrideShellWidgetClass,
                            parent, (Arg *)0, 0);
 XtSetArg(a[i], XtNlabel, desc);i++;
 XtSetArg(a[i], XtNvalue, val);i++;
 dialog = XtCreateManagedWidget("Dialog", dialogWidgetClass, shell, a, i);
 return dialog;
}

/*PUBLIC*/
void PlaceDialog(w, parent)
    Widget w, parent;
{
 int x, y;
 Window child;
 XEvent ev;
 Dimension width, height;
 Arg args[2];
 int i;

 XTranslateCoordinates(dpy, XtWindow(parent), root, 0, 0, &x, &y, &child);
 XtRealizeWidget(XtParent(w)); /* To get the widget's width/height set */
 i=0;
 XtSetArg(args[i], XtNwidth, &width);i++;
 XtSetArg(args[i], XtNheight, &height);i++;
 XtGetValues(w, args, i);
 if (x < 0) x = 0;
 if (x + width > XDisplayWidth(dpy, screen))
   x = XDisplayWidth(dpy, screen) - width;
 if (y < 0) y = 0;
 if (y + height > XDisplayHeight(dpy, screen))
   y = XDisplayHeight(dpy, screen) - height;
 XtMoveWidget(XtParent(w), x, y);
 XtPopup(XtParent(w), XtGrabNone);
 while (XCheckWindowEvent(dpy, XtWindow(parent), ButtonPressMask, &ev)) ;
}

static Widget UserSelectWidget()
{
 XEvent ev;
 XButtonEvent *bptr = (XButtonEvent *) &ev;
 Widget widget;
 static Cursor cursor = NULL;
 if (!cursor) cursor = XCreateFontCursor(dpy,XC_coss);
 if (XGrabPointer(dpy, XDefaultRootWindow(dpy), True,
                  ButtonPressMask|ButtonReleaseMask,
                  GrabModeAsync, GrabModeAsync, None,
                  cursor, CurrentTime) == GrabSuccess) {
  XMaskEvent(dpy, (unsigned long) ButtonPressMask, &ev);
  XMaskEvent(dpy, (unsigned long) ButtonReleaseMask, &ev);
  XUngrabPointer(dpy, CurrentTime);
  /* find the shell widget that is an ancestor of this window */
  widget = XtWindowToWidget(dpy, bptr->window);
  if (widget)
    while (!XtIsSubclass(widget, shellWidgetClass))
        widget = XtParent(widget); /* every non-shell widget has a parent */
  return widget;
 } else return (Widget)0;
}

/* these 2 globlics are kludges to update ShowParam (KLUDGED to EVENT.C)*/
Widget paramWidget = NULL;
caddr_t paramWinData = NULL;

/*ARGSUSED*/
static void NumberLineCallback(w, client, call)
Widget w;
caddr_t client, call;
{
 double present;
 int place, type, done;

 NumberLineGetValue(w, &present, &done);
 place = ((int)client) / 100;
 type =  ((int)client) - place*100;

#ifdef DEBUG
 printf("Number Line Callback:   present = %lf\n", present);
#endif
 if (done && w) {
  XtUnmapWidget(XtParent(w));
  NumberLineCancelType(XtParent(w));
  XtDestroyWidget(XtParent(w));
 } else
  ScrollRelim(place, type, present);
}

/*ARGSUSED*/
static void dummyHandler(w, client, event)
    Widget w; caddr_t client; XEvent *event;
{ ; /*nothing*/
}

static Widget CreateAppWidget(name, class, shellFct, args, argCount)
    char *name;
    WidgetClass class;
    Widget *shellFct; /*RETURN*/
    ArgList args;
    int argCount;
{
 Widget w, shell;
 Arg largs[10];
 int i = 0;
 XtSetArg(largs[i], XtNallowShellResize, True);i++;
 XtSetArg(largs[i], XtNiconName, name);i++;
#ifdef XAW_BC
 shell = XtCreateApplicationShell(name,topLevelShellWidgetClass, largs, i);
#else /* ! XAW_BC */
 shell = XtAppCreateShell(NULL, "CVsim",topLevelShellWidgetClass,dpy,largs,i);
#endif /* XAW_BC */
 w     = XtCreateManagedWidget(name, class, shell, args, argCount);
 XtAddEventHandler(shell, ButtonPressMask | ButtonReleaseMask, False,
                   dummyHandler, (caddr_t)0);
 XtRealizeWidget(shell);
```

```
  if (shellRet) *shellRet = shell;
  return(w);
}

/*ARGSUSED*//*PUBLIC*/
void KillDialog(w, client, call)
Widget w;
caddr_t client, call;
{
  XEvent ev;
  if (client) {
    XtUnmapWidget(XtParent((Widget) client));
    XtDestroyWidget(XtParent((Widget) client));
  }
  while (XCheckWindowEvent(dpy, XtWindow(w), ButtonPressMask, &ev)) ;
}

/*
 *
 * Button Action Function Definitions (private)
 *
 */

/*ARGSUSED*/
static void CloseSeries(w, client, call)
Widget w;
caddr_t client, call;
{
  int *series_ptr = (int *) client;
  (*series_ptr)++;
}

/*ARGSUSED*/
static void Help(widget, client, call)
Widget widget;
caddr_t client, call;
{
  Widget shell;
  Arg args[10];
  int i = 0;
  char buf[256];

  if (CVLib) (void) sprintf(buf, "%s/help.txt", (char *) CVLib);
  else       (void) strcpy(buf, "/mit/hst/cvLib/help.txt");


#ifdef XAW_BC
  XtSetArg(args[i], XtNwidth, 700);i++;
  XtSetArg(args[i], XtNheight, 570);i++;
  XtSetArg(args[i], XtNresizable, False);i++;
  shell = XtCreateApplicationShell("Help", topLevelShellWidgetClass,
                                   (ArgList)0, 0);
  XtSetArg(args[i], XtNfile, buf);i++;
  XtSetArg(args[i], XtNiconName, "Help");i++;
  XtCreateManagedWidget("help", asciiDiskWidgetClass, shell, args, i);
#else /* ! XAW_BC */
  XtSetArg(args[i], XtNiconName, "Help");i++;
  shell = XtAppCreateShell(NULL,"CVsim",topLevelShellWidgetClass,
                           dpy,args, i);

  XtSetArg (args[i], XtNtype, XawAsciiFile); i++;
  XtSetArg (args[i], XtNstring, buf); i++;
  XtSetArg(args[i], XtNwidth, 700);i++;
  XtSetArg(args[i], XtNheight, 570);i++;
  XtSetArg(args[i], XtNresizable, False);i++;
  XtCreateManagedWidget("help", asciiTextWidgetClass, shell, args, i);
#endif /* XAW_BC */
  XtAddEventHandler(shell, ButtonPressMask | ButtonReleaseMask, False,
                    dummyHandler, (caddr_t)0);
  XtRealizeWidget(shell);
  XStoreName(dpy, XtWindow(shell), "Help with Cardiovascular Simulator");
}

/*should be static but event.c kludge*/
/*ARGSUSED*/
void ShowParameters(w, client, call)
Widget w;
caddr_t client, call;
{

  CVData data = (CVData) client;
  Arg args[10];
  int i = 0;
  SIMULATION *present, *normal, s;
  char *formatted = NULL;
  XEvent ev;
#ifndef XAW_BC
  XFontStruct *paramFont;
#endif

  if (data->patientActive) {
    if (paramWidget) {
      if (XtParent(paramWidget)) XtDestroyWidget(XtParent(paramWidget));
      else XtDestroyWidget(paramWidget);
      paramWidget = NULL;
    }
    return; /* Don't look at patient parameters */
  }
  present = presentsim(&s);
  normal = getsimulation(NORMAL);
  formatted = FormatParams(present, normal);
  i=0;
  XtSetArg(args[i],XtNstring, formatted);i++;
#ifdef XAW_BC
  XtSetArg(args[i],XtNeditType,XawtextEdit);i++;
#else /* ! XAW_BC */
  XtSetArg(args[i],XtNeditType,XawtextEdit);i++;
  paramFont = XLoadQueryFont (dpy, "Fixed");
  XtSetArg(args[i], XtNfont, paramFont); i++;
#endif /* XAW_BC */
  XtSetArg(args[i],XtNborderWidth, 2);i++;
  if (!paramWidget) {
    Widget shell;
#ifdef XAW_BC
    paramWidget = CreateAndPopupWidget("Parameters", asciiStringWidgetClass,
                                       &shell, args, i);
#else /* ! XAW_BC */
    paramWidget = CreateAndPopupWidget("Parameters", asciiTextWidgetClass,
                                       &shell, args, i);
#endif /* XAW_BC */
    paramWinData = (caddr_t) data;
    XtResizeWidget(shell, 400, 250, 2);
```

```
  } else {
    Widget shell = XtParent(paramWidget);
    XtDestroyWidget(paramWidget);
#ifdef XAW_BC
    paramWidget = XtCreateManagedWidget("Parameters", asciiStringWidgetClass,
                                        shell, args, i);
    replaceText(paramWidget, formatted);
#else /* ! XAW_BC */
    paramWidget = XtCreateManagedWidget("Parameters", asciiTextWidgetClass,
                                        shell, args, i);
#endif /* XAW_BC */
    paramWinData = (caddr_t) data;
  }
  if (w) while (XCheckWindowEvent(dpy, XtWindow(w), ButtonPressMask, &ev)) ;
}

#ifdef NOTDEFYET
char *FormatVariables(present)
SIMULATION *present;
{
  return("Not yet ready, sorry.\n");
}

static void Summary(widget, client, call)
Widget widget;
caddr_t client, call;
{

  static Widget varWidget = NULL;
  CVDataPtr data = (CVData) client;
  Arg args[10];
  int i = 0;
  SIMULATION *present, *;
  char *formatted = NULL;
  XEvent ev;
#ifndef XAW_BC
  XFontStruct *paramFont;
#endif

  present = presentSim(&);
  formatted = FormatVariables(present);i++;
  i=0;
  XtSetArg(args[i],XtNstring, formatted);i++;
#ifdef XAW_BC
  XtSetArg(args[i],XtNeditType,XttextEdit);i++;
#else /* ! XAW_BC */
  XtSetArg(args[i],XtNeditType,XawtextEdit);i++;
  paramFont = XLoadQueryFont (dpy, "Fixed");
  XtSetArg (args[i], XtNfont, paramFont); i++;
#endif /* XAW_BC */
  XtSetArg(args[i],XtNborderWidth, 2);i++;
  if (!varWidget) {
    Widget shell;
#ifdef XAW_BC
    varWidget = CreateInfoWidget("Parameters", asciiStringWidgetClass,
                                 &shell, args, i);
#else /* ! XAW_BC */
    varWidget = CreateInfoWidget("Parameters", asciiTextWidgetClass,
                                 &shell, args, i);
#endif /* XAW_BC */
    XtRealizeWidget(shell, 400, 250, 2);
  } else {
    replaceText(varWidget, formatted);
  }
  if (widget)
    while (XCheckWindowEvent(dpy, XtWindow(widget), ButtonPressMask, &ev)) ;

}
#endif NOTDEFYET

/*ARGSUSED*/
static void ExitProgram(w, client, call)
Widget w;
caddr_t client, call;
{
#ifdef DEBUG
  printf("Exiting.\n");
#endif
  printf("Thank you for using the Cardiovascular Simulator.      Bye!\n\n");
  exit((int) client); /* now let this process die with the given value */
}

/*ARGSUSED*/
static void Quit(widget, client, call)
Widget widget;
caddr_t client, call;
{
  Widget w, yes, no;
  static int modal_semaphore = 1;

  if (modal_semaphore) {
    modal_semaphore--;
    w = DialogCreate(widget, "Are you sure?", (char *)NULL);
    yes = XtCreateManagedWidget("Quit", commandWidgetClass, w,
                                (Arg *)NULL, 0);
    XtAddCallback(yes, XtNcallback, ExitProgram, (caddr_t) 0);
    XtAddCallback(yes, XtNcallback, CleanSema, (caddr_t) &modal_semaphore);
    no = XtCreateManagedWidget("Continue", commandWidgetClass, w,
                               (Arg *)NULL,0);
    XtAddCallback(no,  XtNcallback, KillDialog, (caddr_t) w);
    XtAddCallback(no,  XtNcallback, CleanSema,  (caddr_t) &modal_semaphore);
    PlaceDialog(w, widget);
  }
}

static Widget toggleDialogBox = NULL;
/*ARGSUSED*/
static void StartTrans(w, client, call)
Widget w;
caddr_t client, call;
{
  CVData data = (CVData) client;
  Arg args[2];
  XEvent ev;

  ResumeSimulator();
  XtSetArg(args[0], XtNlabel, "Stop (running)");
  XtSetValues(data->buttons[STOPBUTTON], args, 1);
  data->simulating = data->simulating;
  KillDialog(toggleDialogBox,(caddr_t)toggleDialogBox, (caddr_t)NULL);
  toggleDialogBox = NULL;
  while (XCheckWindowEvent(dpy, XtWindow(w), ButtonPressMask, &ev)) ;
```

```
}

/*ARGSUSED*/
static void StartSteady(w, client, call)
Widget w;
caddr_t client, call;
{
 CVData data = (CVData) client;
 Arg args[2];
 SIMULATION p;
 XEvent ev;

 ResetSimulator(presentsim(&p));
 XtSetArg(args[0], XtNlabel, "Stop  (running)");
 XtSetValues(data->buttons[STOPBUTTON], args, 1);
 data->simulating = !data->simulating;
 KillDialog(toggleDialogBox,(caddr_t)toggleDialogBox, (caddr_t)NULL);
 toggleDialogBox = NULL;
 while (XCheckWindowEvent(dpy, XtWindow(w), ButtonPressMask, &ev)) ;
}

/*ARGSUSED*/
static void SimToggleOnOff(widget, client, call)
Widget widget;
caddr_t client, call;
{
 CVData data = (CVData) client;
 XEvent ev;

 if (toggleDialogBox) {
  if (data->simulating) {
   Arg args[1];
   StopSimulator();
   XtSetArg(args[0], XtNlabel, "Resume  (stopped)");
   XtSetValues(data->buttons[STOPBUTTON], args, 1);
   data->simulating = !data->simulating;
  } else {
   Widget w, no, yes;

   w = DialogCreate(widget, "Reestablish Steady State?",(char *)NULL);
   toggleDialogBox = w;
   no = XtCreateManagedWidget("TRANSIENT", commandWidgetClass, w,
                              (Arg *)NULL, 0);
   yes =XtCreateManagedWidget("STEADY STATE", commandWidgetClass, w,
                              (Arg *)NULL, 0);
   if (mSensorOn)  {
       Arg args[2];
       XtSetArg(args[0], XtNsensitive, False);
       XtSetValues(yes, args, 1);
   }
   XtAddCallback(no, XtNcallback, StartTrans, (caddr_t)data);
   XtAddCallback(yes, XtNcallback, StartSteady, (caddr_t)data);
   PlaceDialog(w, widget);
  }
 }
 while (XCheckWindowEvent(dpy, XtWindow(widget), ButtonPressMask, &ev)) ;
}

/*ARGSUSED*/
static void SimParams(w, client, call)
Widget w;
caddr_t client, call;
{
 CVData data = (CVData) client;
 Widget widget, shell;
 Arg args[10];
 int i = 0, j = 0;
 int place, type;
 VarData *vd;
 double junk[9];
 double present;
 char buf[128];
 XEvent ev;
 Arg largs[10];

 modify_parameters(0, &place, &type);
 if (place < 0 || type < 0) return; /* No choice made, abort */
 vd = get_vardata(place, type);
 present = getval(junk, place, type); /* place, type for indep. var */

 if (!data->patientActive)
   (void) sprintf(buf, "%s  (%s)", vd->name, vd->units);
 else
   (void) sprintf(buf, "%s  (%% of original)", vd->name, vd->units);
 i=0;
 XtSetArg(args[i], XtNlabel, strcpy(XtMalloc(strlen(buf)), buf));i++;
 XtSetArg(args[i], XtNminimum, &vd->min);i++;
 XtSetArg(args[i], XtNmaximum, &vd->max);i++;
 XtSetArg(args[i], XtNpresent, &present);i++;
 XtSetArg(args[i], XtNdefault, &vd->normal);i++;
 XtSetArg(args[i], XtNprevious, &present);i++;
 if (data->patientActive) {
  XtSetArg(args[i], XtNpercentage, 1);i++;
 } else {
  XtSetArg(args[i], XtNpercentage, 0);i++;
 }

 j = 0;
 XtSetArg(largs[j], XtNallowShellResize, True);j++;
 XtSetArg(largs[j], XtNiconName, vd->name);j++;
#ifdef XAW_BC
 shell = XtCreateApplicationShell(vd->name, topLevelShellWidgetClass,
                                  largs, j);
#else /* !XAW_BC */
 shell = XtAppCreateShell(NULL, "CVsim",topLevelShellWidgetClass,dpy,largs,j);
#endif /* XAW_BC */
 widget = XtCreateManagedWidget(vd->name, numberLineWidgetClass,
                                shell, args, i);
 XtAddEventHandler(shell, ButtonPressMask | ButtonReleaseMask, False,
                   dummyHandler, (caddr_t)0);
 if (NumberLineCheckType(place,type)) {
  /* disallow 2 manipulations of same place&type */
  NumberLineCancelType(shell);
  XtDestroyWidget(shell);
  return;
 }
 XtRealizeWidget(shell);
 (void) NumberLineResetAll(shell, place, type);
 XSetIconName(dpy, XtWindow(shell), vd->name);
 XtAddCallback(widget, XtNcallback, NumberLineCallback,
               (caddr_t)(place*100 + type));
 while (XCheckWindowEvent(dpy, XtWindow(w), ButtonPressMask, &ev)) ;
```

```
}

/*
 * The following procedures deal with the form for
 * creating new plots.
 */


/*ARGSUSED*/
static void MakePlot(w, args, argCount)
Widget w;
ArgList args;
int argCount;
{
  Widget widget, shell;
  XEvent ev;

  widget = CreateIndexWidget("Plot",plotWidgetClass,&shell, args, argCount);
  if (widget) AddDataWidget(widget, (XtCallbackProc)0, PlotData);
  while (XCheckWindowEvent(dpy, XtWindow(w), ButtonPressMask, &ev)) ;
}

/*ARGSUSED*/
static void NewPlot(widget, client, call)
Widget widget;
caddr_t client, call;
{
  Widget menu;
  XEvent ev;
  if(menu = MakePlotMenu(widget, MakePlot, 500, 5,
                         (AxisData *)NULL, (AxisData *)NULL,
                         (AxisData *)NULL, (AxisData *)NULL))
    XtPopup(menu, XtGrabExclusive);
  while (XCheckWindowEvent(dpy, XtWindow(widget), ButtonPressMask, &ev)) ;
}

/*PUBLIC*/ Widget BUTTON_BOX;
static Widget HisShell;

/*
 * Generic widget deletor
 */
/*ARGSUSED*/
static void DeletePlot(widget, client, call)
  Widget widget;
  caddr_t client, call;
{
  Widget wid;
  XEvent ev;

  wid = UserSelectWidget();
  if (wid && BUTTON_BOX && wid != XtParent(BUTTON_BOX)) {
    DeleteDataWidget(wid);
    NumberLineCancelType(wid);
    XtUnmapWidget(wid);
    XtDestroyWidget(wid);
    if (paramWidget && (wid == XtParent(paramWidget))) paramWidget = NULL;
    if (HisShell && (wid == HisShell || wid == XtParent(HisShell)))HisShell = NULL;
  }
  while (XCheckWindowEvent(dpy, XtWindow(widget), ButtonPressMask, &ev)) ;
}

#ifdef NOTDEFYET
struct pixlist {
  Pixmap pixmap;
  struct pixlist *next;
} ;

static void Save(widget, client, call)
    Widget widget;
    caddr_t client, call;
{
  Widget wid;

  wid = UserSelectWidget();
  if (wid && BUTTON_BOX && wid != XtParent(BUTTON_BOX)) {

  }
}
#endif NOTDEFYET

/*ARGSUSED*/
static void PrintPlot(widget, client, call)
Widget widget;
caddr_t client, call;
{
  char *printCmd;
  XEvent ev;
  int s;

  printCmd = getenv("CVPRINT");
  if (!printCmd || strlen(printCmd) < 2)
    printCmd = "xdpr";
  if (s=system(printCmd))
    fprintf(stderr, "%s:    exit status %d",printCmd, s);
  while (XCheckWindowEvent(dpy, XtWindow(widget), ButtonPressMask, &ev)) ;
}

static Widget resetWidget;

/*ARGSUSED*/
static void ResetAction(widget, client, call)
Widget widget;
caddr_t client, call;
{
  CVData data = (CVData) client;
  char buf[1024];
  XEvent ev;

  (void) strcpy(buf, CVLib);
  (void) strcat(buf, "/parameter.db");
  LoadParameterData(buf);
  ReInitCVSIM();
  if (data->printActive)
    NumberLineDestroyAll();
  else
    NumberLineResetAll();
  if (paramWidget)
    ShowParameters(paramWidget, (caddr_t) paramWinData, (caddr_t)9);
  if (HisShell) XtDestroyWidget(HisShell); HisShell = 0;
  ResetSimulator(setsimulation());
  if (!data->simulating) StopSimulator();
```

```
    data->patientActive=0;
    patientActive = 0;
    {
     Arg args[2]; /* MAKE PARAMETER LIST BUTTON SENSITIVE */
     XtSetArg(args[0], XtNsensitive, True);
     XtSetValues((Widget) data->buttons[LISTBUTTON], args, 1);
    }
    KillDialog(resetWidget, (caddr_t)resetWidget, (caddr_t)NULL);
    while (XCheckWindowEvent(dpy, XtWindow(widget), ButtonPressMask, &ev)) ;
  }

  /*ARGSUSED*/
  static void ResetButton(widget, client, call)
  Widget widget;
  caddr_t client, call;
  {
   CVData data = (CVData) client;
   Widget w, reset, cancel;

   resetWidget = w =
     DialogCreate(widget,"This will reset the system.",(char **)NULL);
   reset = XtCreateManagedWidget("Reset", commandWidgetClass,w, (Arg *)0,0);
   cancel= XtCreateManagedWidget("Cancel", commandWidgetClass,w, (Arg *)0,0);
   XtAddCallback(reset, XtNcallback, ResetAction, (caddr_t)data);
   XtAddCallback(cancel, XtNcallback, KillDialog, (caddr_t)w);
   PlaceDialog(w, widget);
  }

  static void ShowHx(history)
  char *history;
  {
  #ifdef XAW_BC
    Arg args[5];
  #else
    Arg args[10];
    XFontStruct *historyFont;
  #endif
    int i=0;

    /* Show only 1 history at a time. */
    if (HxShell) XtDestroyWidget(HxShell); HxShell = (Widget) 0;
    XtSetArg(args[i], XtNstring, history);i++;
    XtSetArg(args[i], XtNeditType, XawtextRead);i++;
  #ifdef XAW_BC
    XtSetArg(args[i], XtNresizable, True);i++;
    CreateDialogWidget("History", asciiStringWidgetClass, &HxShell, args, i);
  #else /* ! XAW_BC */
    /*
     * The history popup was not resizing to accommodate the text --
     * hence the given width and height resources.  Probably should be
     * fixed someday.  dlb 8/10/90
     */
    historyFont = XLoadQueryFont (dpy, "fixed");
    XtSetArg(args[i], XtNtype, XawAsciiString); i++;
    XtSetArg(args[i], XtNdisplayCaret, True); i++;
    XtSetArg (args[i], XtNlength, strlen(history)); i++;
    XtSetArg(args[i], XtNeditType, XttextReadWrite);i++;
    XtSetArg(args[i], XtNfont, historyFont); i++;
    XtSetArg(args[i], XtNwidth, 500); i++;
    XtSetArg(args[i], XtNheight, 225); i++;
    CreateDialogWidget("History", asciiTextWidgetClass, &HxShell, args, i);
  #endif /* XAW_BC */
  }

  /*ARGSUSED*/
  static void callbackPopdown(w, client, call)
  Widget w;
  caddr_t client, call;
  {
   XtPopdown((Widget)client);
  }

  typedef struct {
    CVData data;
    char *path;
  } PatientDataRec, *PatientData;

  static char KEYWORD[80];

  /*ARGSUSED*/
  static void DoPatient(widget, client, call)
  Widget widget;
  caddr_t client, call;
  {
   CVData data = ((PatientData) client)->data;
   char *path =  ((PatientData) client)->path;
   char *history;
   char buf[10024];
   XEvent ev;
   Widget parent = XtParent(widget);

   if (parent) callbackPopdown(widget, (caddr_t) XtParent(parent), call);
   (void) strcpy(buf, CVLib);
   (void) strcat(buf, "/parameter.db");
   LoadParameterData(buf);
   ReframeOS();
   (void) sprintf(buf, "crypt %s < %s > /tmp/pt.pov01", KEYWORD, path);
   (void) system(buf);
   if ((history = LoadPatient("/tmp/pt.pov01")) == NULL) {
    (void) unlink("/tmp/pt.pov01");
    return; /* no such patient file */
   }
   (void) unlink("/tmp/pt.pov01");
   data->patientActive = 1;
   patientActive = 1;
   {
    Arg args[2]; /* MAKE PARAMETER LIST BUTTON INSENSITIVE */
    XtSetArg(args[0], XtNsensitive, False);
    XtSetValues((Widget) data->buttons[LISTBUTTON], args, 1);
   }
   ResetSimulator(estimulation());
   NumberLineDestroyAll();
   if (paramWidget) {
    XtDestroyWidget(XtParent(paramWidget));
    paramWidget = NULL;
   }
  if (data->simulating) StopSimulator();
  #ifdef DEBUG
   printf("%s\n",history);
  #endif
   ShowHx(history);
   XtPopdown(data->patient);
```

```
    while (XCheckWindowEvent(dpy, XtWindow(widget), ButtonPressMask, &ev)) ;
}

/*ARGSUSED*/
static void Patients(widget, client, call)
Widget widget;
caddr_t client, call;
{
  CVData data = (CVData) client;
  Widget buttons, cancel;
  static Widget box;
  int i;
  int fd, n=0;
  Arg args[10];
  char buf[1024], name[1024];
  PatientData patientData;

  if (!data->patients) {
    data->patients = XtCreatePopupShell("PatientsShell",
                                        overrideShellWidgetClass,
                                        widget, (Arg *)NULL, 0);
    box = XtCreateManagedWidget("patientList", boxWidgetClass,
                                data->patients, (Arg *)NULL, 0);
    i=0;
    XtSetArg(args[i], XtNborderWidth, 0);i++;
    XtCreateManagedWidget("Patient Unknowns", labelWidgetClass, box, args, i);
    for (n=0; n<20; n++) {
      (void) sprintf(buf,"%s/Patients/hist.%1d",CVLib,n);
      if ((fd=open(buf,O_RDONLY,0)) > 0) {
          (void) close(fd);
          i=0;
          (void) sprintf(name,"Case %9d",n);
#ifdef DEBUG
          printf("%s\n",name);
#endif
          XtSetArg(args[i], XtNlabel,
                   strcpy(XtMalloc(strlen((char*)strlen(name)), name));i++;
          button =XtCreateManagedWidget("patient",commandWidgetClass,box,args,i);
          patientData = (PatientData) XtMalloc(sizeof(PatientDataRec));
          patientData->data = data;
          patientData->path = strcpy(XtMalloc(strlen((char*)strlen(buf)),buf);
          XtAddCallback(button, XtNcallback, DrPatient, (caddr_t)patientData);
      }
    }
    cancel = XtCreateManagedWidget("Cancel", commandWidgetClass, box,
                                   (Arg *)0,0);
    XtAddCallback(cancel, XtNcallback, callbackPopdown,(caddr_t)XtParent(box));
  }
  PlaceDialog(box, widget);
}

/*
 *
 * Public definitions.
 *
 */
static XContext nlContext = NULL;

/*PUBLIC*/ /*ARGSUSED*/
Widget InstanceCreate(parent, args, argCount)
Widget parent;                          /* Window to put the form in. */
ArgList args;                           /* Args to pass on to form (if any). */
int argCount;
{
  CVData data = (CVData) XtMalloc(sizeof(CVDataRec));
  register int i;

  { /* smurf keyword for later crypt calls */
    FILE *kwfile;
    char buf[1024];

    (void) sprintf(buf, "crypt keyword < %s/Patients/kw.cr > /tmp/pt.pck01",
                   CVLib);
    (void) system(buf);
    kwfile = fopen("/tmp/pt.pck01","r");
    (void) fgets(KEYWORD, 79, kwfile);
    (void) fclose(kwfile);
    (void) unlink("/tmp/pt.pck01");
    KEYWORD[strlen(KEYWORD)-1] = '\0'; /* make the stupid \n from fgets */
  }
  if (nlContext == NULL) nlContext = XUniqueContext();
  { /* start with a help screen*/
    Widget w;
    Arg args[10];
    int i = 0;
    char buf[256];

    if (CVLib) (void) sprintf(buf, "%s/help.txt", (char *) CVLib);
    else       (void) strcpy(buf, "help.txt");

#ifdef XAW_BC
    w = XtCreateApplicationShell("Help", transientShellWidgetClass,
                                 (ArgList)0, 0);
    XtSetArg(args[i], XtNfile, buf);i++;
    XtSetArg(args[i], XtNiconName, "Help");i++;
    XtCreateManagedWidget("Help", asciiDiskWidgetClass, w, args, i);
#else /* ! XAW_BC */
    XtSetArg(args[i], XtNiconName, "Help");i++;
    w = XtAppCreateShell(NULL,"CVsim",transientShellWidgetClass,
                         dpy,(Arg *)args,i);

    XtSetArg (args[i], XtNtype, XawAsciiFile); i++;
    XtSetArg (args[i], XtNstring, buf); i++;
    XtSetArg(args[i], XtNwidth, 780);i++;
    XtSetArg(args[i], XtNheight, 570);i++;
    XtSetArg(args[i], XtNscrollVertical, False);i++;
    XtCreateManagedWidget("Help", asciiTextWidgetClass, w, args, i);
#endif /* XAW_BC */
    XtAddEventHandler(w, ButtonPressMask | ButtonReleaseMask, False,
                      dummyEventHandler, (caddr_t)0);
    XtRealizeWidget(w);
    XStoreName(dpy, XtWindow(w), "Help with Cardiovascular Simulator");
  }
  {
    Widget w;
    Arg args[5];
    int i = 0;
    char *name = "Simulator Command Buttons";

    XtSetArg(args[i], XtNiconName, name);i++;
#ifdef XAW_BC
    w = XtCreateApplicationShell(name,topLevelShellWidgetClass, args, i);
```

```
#else /* ! XAW_3D */
    w = XtAppCreateShell(NULL,"CVsim",applicationShellWidgetClass, dpy, args, i);
#endif /* XAW_3D */
    data->buttonBox = XtCreateManagedWidget("commandButtons", boxWidgetClass,w,
                                            (Arg *)0,0);
    BUTTON_BOX = data->buttonBox; /* to be saved during spring cleaning */
    XtConfigureWidget(w, XDisplayWidth(dpy,XDefaultScreen(dpy))-240-100,
                      10, 230, 125, 4);
    XtRealizeWidget(w);
    XStoreName(dpy, XtWindow(w), "CV Simulator Buttons");
  }
  data->simulating = TRUE;
  data->patientActive = 0;
  patientActive = 0;
  data->administrator = 0;
  data->patients = NULL;
  for (i=0; Buttons[i]; i++) {
    static char *trans = "<BtnUp>:    notify() unset() unhighlight() \n";
    Arg args[1];
    int count = 0;
    XtSetArg(args[0], XtNlabel, Buttons[i]);count++;
    data->buttons[i] =XtCreateManagedWidget("commandButton",
                                            commandWidgetClass,
                                            data->buttonBox, args, count);
    XtOverrideTranslations(data->buttons[i], XtParseTranslationTable(trans));
    XtAddCallback(data->buttons[i], XtNcallback, ButtonProc[i], (caddr_t)data);
  }
#ifdef REFLEX_OFF
  {
    Arg args[2]; /* MAKE BARORECEPTOR BUTTON INSENSITIVE */
    XtSetArg(args[0], XtNsensitive, False);
    XtSetValues((Widget) data->buttons[REFLEXBUTTON], args, 1);
  }
#endif REFLEX_OFF
  return ((Widget)0);
}
```

Main.c

```
#ifndef lint
#include "copyright.h"
static char *sccs="@(#)andor:    /mit/hst/src/main/RCS/Main.c,v 2.5 90/08/28 14:51:49 tldavis Exp";
#endif
/**************
 * $Source: /mit/hst/src/main/RCS/Main.c,v $
 * Cardiovascular Simulator
 * Copyright 1990 Timothy L. Davis
 *
 * This routine gets the show on the road by installing the
 * parameter and patient databases, and creating an instance of the
 * user interface.
 * The main control loop is contained in
 * ../event/Event.c::DispatchEventsAndSignals().
 * $Log:            Main.c,v $
 * Revision 2.5  90/08/28  14:51:49  tldavis
 * Hmmm.
 *
 * Revision 2.3  90/08/18  18:46:57  tldavis
 * General cleanup, using X11R4-style setup.
 *
 * Revision 2.2  90/08/10  10:31:43  tld
 * R4 updates; Xaw includes are now in X11/Xaw
 *
 * Revision 2.1  90/08/07  16:19:13  tldavis
 * Removing unused vars & functions from compilation.
 *
 * Revision 2.0  90/08/06  17:56:19  tldavis
 * Final X11R3 version.
 *
 * Revision 1.7  90/08/06  17:07:09  tldavis
 * Added some epicurial color code from Dr. Sanza.
 *
 * Revision 1.6  90/05/31  16:56:10  tldavis
 * Pre-ACES checkin.
 *
 * Revision 1.5  90/05/28  11:52:51  tldavis
 * X11R3 Version, no baroreceptor yet.
 *
 * Revision 1.4  88/10/11  01:55:04  hst
 * removed DEBUG flag.
 *
 * Revision 1.3  88/10/10  01:29:02  tldavis
 * Fixed loss of logs.  A few remain(on paper).
 *
 * Revision 1.2  88/10/05  11:18:55  tldavis
 * Made button box pop-up.  Need to fix hints to place on screen.
 *
 * Revision 1.1  88/10/04  10:44:16  tldavis
 * Initial revision
 *
 **************/
/* For Dr. Sanza; should be included by Intrinsic.h anyway. */
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
/* End of includes for Dr. Sanza */
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Shell.h>
#include <math.h>
#include <strings.h>
#include "../event/Event.h"
#include "../plot/Plot.h"
#include "../ml/NumberLine.h"
#include "Interface.h"
#include "../sim/CVDefs.h"
#include "../data/Parameter.h"

extern char *getenv();
extern void BellmanOff();
/*PUBLIC*/ char *CVLib; /* location of library files */
/*PUBLIC*/ Display *dpy; int screen; Window root;
/*PUBLIC*/ unsigned long white_pixel, black_pixel, border_pixel;

main(argc, argv)
    int argc;
    char **argv;
```

```
{
  Arg args[10];
  int i;
  char buf[256];
#ifndef XAW_BC
  XtnOptionDescRec options[2];
  XtAppContext app_context;
#else XAW_BC
  Widget shell;
#endif XAW_BC

  if (!(CVLib = getenv("CVLIB"))) CVLib = "/mit/hst/cvLib";
  sprintf(buf, "%s/parameter.db",CVLib);
  LoadParameterDatabase(buf);
  ReleaseOSI();
  setsimulation();

#ifdef DEBUG
  printf("XtAppInitialize...\n");
#endif
#ifdef XAW_BC
  shell = XtInitialize(argv[0],"CVsim", (XtnOptionDescRec *)NULL,
                       (Cardinal)0, (Cardinal *)&zero, (String *)argv);
  dpy = XtDisplay(shell);
#else !XAW_BC X11R4 stuff
  XtToolkitInitialize();
  app_context = XtCreateApplicationContext();
  dpy = XtOpenDisplay(app_context, (String)NULL,
                      (String)NULL, "CVsim",
                      options, (Cardinal)0,
                      (Cardinal *)&zero, (String *)argv);
#endif XAW_BC
  screen = DefaultScreen(dpy);
  root = RootWindow(dpy, screen);

  white_pixel = WhitePixel(dpy, screen);
  black_pixel = BlackPixel(dpy, screen);

#ifdef DEBUG
  XSynchronize(dpy, 1); /* makes it slow but easier for bug tracking */
  printf("Creating Widgets...\n");
#endif

  i=0;
  InterfaceCreate((Widget)0, args, i);
  MakeSimulate(getsimulation(NORMAL),(double)0.0);

  DispatchDataAndEvents(); /* should never return */
}
```

C.2 Mathematical simulation routines

These program files encode the state equations, the Runge-Kutta integration method, and the baroreflex control system with its user interface objects.

CVDefs.h

```
/*
 *      $Source: /mit/hst/src/sim/RCS/CVDefs.h,v $
 *      $Header: /mit/hst/src/sim/RCS/CVDefs.h,v 2.0 90/08/06 17:54:51 tldavis Stab Locker: tldavis $
 */

/*
 *      $Source: /mit/hst/src/sim/RCS/CVDefs.h,v $
 *      $Header: /mit/hst/src/sim/RCS/CVDefs.h,v 2.0 90/08/06 17:54:51 tldavis Stab Locker: tldavis $
 */

/* file: cvdefs.h      Tim Davis, G. Moody, R. Sah
 *                     Copyright 1988 Timothy L. Davis

Constants, macros, and typedefs for CV simulator

Dimensions of variables:

        Quantity        Units
        --------- -----
        pressure        mmHg
        capacitance     ml/mm Hg
        resistance      mm Hg--sec/ml
        volume          ml
        flow            ml/sec
*/
/* These cause a compiler error... endless loop "too m.ch pushback"
extern Display *dpy;
#define screen      DefaultScreen(dpy)
#define white_pixel WhitePixel(dpy, screen)
#define black_pixel BlackPixel(dpy, screen)
#define root        DefaultRootWindow(dpy)
*/

#include <stdio.h>
#include <math.h>

#define  RWXOD          000644
#define MINTSTP         .003999999
#define  SYSTOLESBOS    528     /* # table segments during systole */
#define  TBLSBOS        792     /* 3/2 SYSTOLESBOS */
#define  TBLSBOSPLUS2   TBLSBOS+2

#define  xpl        x[0]            /* left heart pressure */
#define  xpa        x[1]            /* peripheral arterial pressure */
#define  xpv        x[2]            /* peripheral venous pressure */
#define  xpr        x[3]            /* right heart pressure */
#define  xpPa       x[4]            /* pulmonary arterial pressure */
#define  xpPv       x[5]            /* pulmonary venous pressure */
#define xz      x[6]
#define xCl     x[7]
#define xCr     x[8]
#define  dpldt      dxdt[0]         /* left heart pressure */
#define  dpadt      dxdt[1]         /* peripheral arterial pressure */
#define  dpvdt      dxdt[2]         /* peripheral venous pressure */
#define  dprdt      dxdt[3]         /* right heart pressure */
#define  dpPadt     dxdt[4]         /* pulmonary arterial pressure */
#define  dpPvdt     dxdt[5]         /* pulmonary venous pressure */

#define  NORMAL     1

#ifndef TRUE
```

```
#define FALSE           0
#define TRUE            1
#endif TRUE

#define  TIME           0     /* type codes for getval(), setval(), etc*/
#define  FLOW           1
#define  PRESSURE       2
#define  VOLUME         3
#define CAPACITANCE     4
#define CAPACITANCE_DIAS 5
#define CAPACITANCE_SYS 6
#define ZPVOLUME        7
#define RESISTANCE      8
#define HEARTRATE       9
#define ENDVAR  (-1)

#define RESISTOR 0 /* Device codes for user interface */
#define CAPACITOR 1
#define VARCAPACITOR 2
#define SYSTEMOR 3 /* This is a dummy component holding non-localizable vars */

#define DEPENDENT 0 /* variable types */
#define INDEPENDENT 1

#define LV          0    /* place codes for getval(), setval() */
#define SA          1
#define SV          2
#define RV          3
#define PA          4
#define PV          5
#define SC          6
#define PC          7
#define SYSTEM      8

#define LVI 9       /*corresponding flow lines */
#define SCI 10
#define SVI 11
#define RVI 12
#define PCI 13
#define PVI 14

#define  NVL0           15.      /* Normal SIMULATION parameter values */
#define  NVR0           15.
#define  NVA0           715.
#define  NVV0           2500.
#define  NVPA0          90.
#define  NVPV0          400.
#define  NBV            3000.

#define  NRA            1.
#define  NRV            0.05
#define  NRPV           0.05
#define  NRLI           0.010
#define  NRLO           0.035
#define  NRRO           0.005

#define NCA             1.6
#define NCV             109.
#define NCPA            4.5
#define NCPV            8.4
#define NCLDIAS         10.0
#define NCLSYS          0.4
#define NCRDIAS         20.0
#define NCRSYS          1.2

#define  NPTH           (-4.0)
#define NHR             72.0

typedef struct simulation_ {
  double VlI;     /* left heart volume at 0 pressure */
  double VrI;     /* right heart volume at 0 pressure */
  double VaI;     /* peripheral arterial volume at 0 pressure */
  double VvI;     /* peripheral venous volume at 0 pressure */
  double VPaI;    /* pulmonary arterial volume at 0 pressure */
  double VPvI;    /* pulmonary venous volume at 0 pressure */
  double bv;      /* total blood volume */

  double Ra;      /* peripheral arterial resistance */
  double Rv;      /* peripheral venous resistance */
  double RPv;     /* pulmonary vascular resistance */
  double Rli;     /* left ventricular inflow resistance */
  double Rlo;     /* left ventricular outflow resistance */
  double Rro;     /* right ventricular outflow resistance */

  double Ca;      /* peripheral arterial capacitance */
  double Cv;      /* peripheral venous capacitance */
  double CPa;     /* pulmonary arterial capacitance */
  double CPv;     /* pulmonary venous capacitance */
  double Clda,    /* left heart diastolic capacitance */
  double Clsys;   /* left heart systolic capacitance */
  double Crda;    /* right heart diastolic capacitance */
  double Crsys;   /* right heart systolic capacitance */

  double pth;     /* intrathoracic pressure */
  double hr;      /* heart rate in beats per minute */
} SIMULATION;

typedef struct var_dam_node_ {
  int var;
  int dep;
  char *name;
  double min, max, nomask;
  double gmin, gmax;
  char *units;
} VarData;
```

eqns.c

```
/* Modified extensively 1986--1989 Timothy L. Davis
 * $Source: /usr/lst/src/sim/BCS/eqns.c,v $
 * $Log:        eqns.c,v $
 * Revision 2.2  90/08/28  14:44:37  tldavis
 * Added function UpdateTiming() to calculate the portion of the cardiac
 * cycle and store it in an unsigned int.
 *
 * Revision 1.1  90/08/19  16:25:52  tldavis
 * Initial revision
 *
 * Revision 2.1  90/08/18  18:39:51  tldavis
 * New location of CVDefs in this directory.
 *
 * Revision 2.0  90/08/06  18:02:38  tldavis
```

```
 * Final X11R3 version.
 *
 * Revision 1.6  90/08/06  17:03:31  tdavis
 * Added a few new dependent variables: atrial pressures and instantaneous
 * ventricular capacitances.
 *
 * Revision 1.5  90/05/31  16:25:16  tdavis
 * Pre-ACIS checkin.
 *
 * Revision 1.4  90/02/20  08:01:55  tdavis
 * Fixed ventricular constant volume bug.
 *
 * Revision 1.3  89/10/12  22:17:01  tdavis
 * Updated Fall, 1989.
 *
 * Revision 1.2  89/05/21  15:23:03  tdavis
 * Works with heart+reflex control in a single process.
 *
 * Threaded X11R3 version, for cpcom, with control system features.
 *
 * Equations within the circulatory system model including:
 *   -  differential ECL equations at each node
 *   -  heart capacitance values as a function of time via a look-up table
 *      (without interpolation).  NOTE: this assumes the diastolic capacitance
 *      is constant, which is only valid at normal ventricular volumes.
 *
 */
#include "CVDefs.h"
#include <stdio.h>
#include <math.h>
#include "../event/Event.h" /* for UpdateParamWindow() */

extern double sigHR, sigR, sigV, sigCl, sigCr; /* from reflex.c */
extern void calcReflexes(); /* reflex.c */

/*
 * 1/3 of systemic arterial capacitance is in the thorax: ATH_FRAC
 */
#define ATH_FRAC 0.333

/* sensitivity variables below are used by reflex.c exclusively */
static double Vlt;          /* left heart volume at 0 pressure */
static double Vrt;          /* right heart volume at 0 pressure */
static double Vat;          /* peripheral arterial volume at 0 pressure */
double Vvt;       /* peripheral venous volume at 0 pressure */
static double VPat;         /* pulmonary arterial volume at 0 pressure */
static double VPvt;         /* pulmonary venous volume at 0 pressure */
static double bv;           /* total blood volume */
double Ra;        /* peripheral arterial resistance */
static double Rv;           /* peripheral venous resistance */
static double RPv;          /* pulmonary vascular resistance */
static double Rli;          /* left ventricular inflow resistance */
static double Rlo;          /* left ventricular outflow resistance */
static double Rro;          /* right ventricular outflow resistance */
double Ca;        /* peripheral arterial capacitance */
static double Cv;           /* peripheral venous capacitance */
static double CPa;          /* pulmonary arterial capacitance */
static double CPv;          /* pulmonary venous capacitance */
static double Cldias;       /* left heart diastolic capacitance */
double Clsys;     /* left heart systolic capacitance */
static double Crdias;       /* right heart diastolic capacitance */
double Crsys;     /* right heart systolic capacitance */
static double pth;          /* intrathoracic pressure */
double hr;        /* heart rate operating point in beats per minute */
static int DLI;             /* left heart inflow diode */
static int DLO;             /* left heart outflow diode */
static int DRI;             /* right heart inflow diode */
static int DRO;             /* right heart outflow diode */

/* The above variables are the signals used for calculations involving
 * hr, Ra, Vv0, Clsys, and Crsys.  The nominal values of hr, Ra, etc.
 * are used as setpoints for the linear control system if reflexes are
 * on.  If reflexes are off, hr etc. are copied directly into sigHR etc.
 */

/*PUBLIC*/
void initparams(p)
SIMULATION *p;
{
 Vlt = p->Vlt;  Vrt = p->Vrt;  Vat = p->Vat;  Vvt = p->Vvt;
 VPat = p->VPat;  VPvt = p->VPvt;  bv = p->bv;
 Ra = p->Ra;Rv = p->Rv;RPv = p->RPv;Rli = p->Rli; Rlo = p->Rlo; Rro = p->Rro;
 Ca = p->Ca;  Cv = p->Cv; CPa = p->CPa; CPv = p->CPv;
 Cldias = p->Cldias; Clsys = p->Clsys; Crdias = p->Crdias; Crsys = p->Crsys;
 pth = p->pth;  hr = p->hr;
}

/*PUBLIC*/
SIMULATION *presentsim(p)
SIMULATION *p;
{
 p->Vlt = Vlt;  p->Vrt = Vrt;  p->Vat = Vat;  p->Vvt = Vvt;  p->VPat = VPat;
 p->VPvt = VPvt;  p->bv = bv;  p->Ra = Ra;  p->Rv = Rv;  p->RPv = RPv;
 p->Rli = Rli;  p->Rlo = Rlo;  p->Rro = Rro;
 p->Ca = Ca;  p->Cv = Cv;  p->CPa = CPa;  p->CPv = CPv;
 p->Cldias = Cldias;  p->Clsys = Clsys;  p->Crdias = Crdias; p->Crsys = Crsys;
 p->pth = pth;  p->hr = hr;
 return(p);
}

/*PUBLIC*/
void
eqns(dxdt, i, x) /* takes x[] pressures, and time index in systole
                  * then returns changes in dxdt[]
                  */
double x[], dxdt[];
int i;
{
        double Cl(), Cs(), dClds(), dCrds(), Clsv();
        double q0, q1, q2, q3, q4, q5; /* These are FLOWs */

        q0 = DLI ? (xpPv - xpl) /Rli  : 0.;
        q1 = DLO ? (xpl - xpa)  /Rlo  : 0.;
        q2 =       (xpa - xpv)  /sigR;
        q3 = DRI ? (xpv - xpr)  /Rv   : 0.;
        q4 = DRO ? (xpr - xpPa) /Rro  : 0.;
        q5 =       (xpPa - xpPv)/RPv;
        dpth = ((pth - xpl)*dClds(i) + q0 - q1)/Cl(i);
        dpa = (q1 - q2)/Ca;
        dpv = (q2 - q3)/Cv;
        dpr = ((pth - xpr)*dCrds(i) + q3 - q4)/Cs(i);
        dpPa = (q4 - q5)/CPa;
```

```
        dpP*dt = (p5 - p0)/CPv;
}

static  struct  ctable      /* Table of heart contraction capacitance */
{
        double  fcn; /* Maximum capacitance for scaling purposes. Min == 0 */
        unsigned long val[TBLSIZ0SPLUS2];
} ctbl = {
filenbase "tblc"
};

static  xtrust  double      /* table of heart contraction cap. derivatives*/
{
        double  fcn; /* Not used. Calculated from ctbl.fcn * ds/dt */
        long val[TBLSIBOSPLUS2];
} dctbl = {
filenbase "tbldocdt"
};

static double
  k2eTsys,    /* time multiplier for Sys. table lookup*/
  kr1,        /* capacitance multiplier for RH systole*/
  kl1,        /* capacitance multiplier for LH systole*/
  kr2,        /*    "        "       "  RH systole dC/dt */
  kl2;        /*    "        "       "  LH   "    "  */

/*PUBLIC*/
double Cins(dt)
    double dt;
{
 return (dt * k2eTsys);
}

/*PUBLIC*/
double Cr(i)
    int i;
{
 if (i >= TBLSIBOS)  return (Cnilns);
 else  return (sigCr + ctbl.val[i] * kr1); /* ctbl.val is LONG! */
}

/*PUBLIC*/
double   Cl(i)
    int i;
{
 if (i >= TBLSIBOS) return (Clilns);
 else return (sigCl + ctbl.val[i] * kl1);
}

/*PUBLIC*/
double   dCrdt(i)
    int i;
{
 if (i >= TBLSIBOS) return (0.);
 else return (dctbl.val[i] * kr2);
}

/*PUBLIC*/
double dClds(i)
    int i;
{
 if (i >= TBLSIBOS) return (0.);
 else return (dctbl.val[i] * kl2);
}

/*PUBLIC*/
double Tsystole(hrate)
    double hrate;
{
 return(0.30*sqrt(60./hrate));
}

/*
 * lookup tables for heart capacitance used from 0 <= t <= xlTsys.
 * k2eTsys is the number of lookup table items per time in seconds.
 * kr1 is scaling factor for right heart capacitance.
 * kr2 is scaling factor for right heart d(capacitance)/dt in cap. per sec.
 * similar for kl1 and kl2.
 */
/*PUBLIC*/
void tcontset()
{
 double Tsys = Tsystole(sigHR); /* time for contraction (seconds) */;
 k2eTsys = SYSTOLEBOS/Tsys;  /* number of data file segs per time*/
 kl1 = (Clslns - sigCl)/ctbl.fcn; /* scaling factor for left heart */
 kl2 = kl1 * k2eTsys;  /* Scaling factor for dcdt.  Table is in
                        * dc/ds (per segment).  Convert dc/dt to cap
                        * per time by chain rule
                        */
 kr1 = (Cnlns - sigCr)/ctbl.fcn; /* scaling factor for right heart */
 kr2 = kr1 * k2eTsys;  /* scaling factor for dcdt on right */
}


/*PUBLIC*/
void initvalves(x)
double x[];
{
        DLI = (xpPv >= xpl);
        DLO = (xpl  >= xpa);
        DRI = (xpv  >= xpr);
        DRO = (xpr  >= xpPa);
}

/*PUBLIC*/
void finvolume(x)
double x[];
{
    /* find deviation from total blood vol.
     * correct at systemic venous reservoir
     */
 double diff = (xv - (VBo+VaBo+sigV+VaDo+VPuBo+VPvO) -
                (xCl*(xpl-pth) + Ca*(xpa-A_TH_FRAC*pth) + Cv*xpv +
                 xCr*(xpr-pth) + CPa*(xpPa-pth) + CPv*(xpPv-pth)))/Cv;
#ifdef DEBUG
 if (fabs(diff) > 1.0)
  printf("Volume error:       %g ml    ",diff);
#endif DEBUG
 xpv += diff;
}

/*PUBLIC*/
```

```
double VenVol(z)
    double z[];
{
 return(zpv * Cv);
}

/* Returns TRUE for valve state within given period if pd1 & pd2 are different
 * Returns TRUE at leading edge of pd1 if pd2 == 0.
 * Sets bit INPERIOD_POS as entrance, clears on exit.
 * FALSE otherwise.
 * BIT CODE:  0, 1:  DLI, DLO beginning desired period, immediately following
              2, 3:  the previous DLI, DLO to bits 0, 1.
              4, 5:  DLI, DLO immediately following the end of the periods, with
              6, 7:  DLI, DLO the last valve state in the period.
              8:     Flag set in this routine 1: in period, 0: not in.
              9, 10: The current flag settings saved here from last time.
              11:    1 if averages are being plotted instead of direct data
              12:    1 at the start of each beat (beginning of diastole)
              13:    1 during diastole
          14-15:     reserved
 * This uses twice as many bits as necessary for the number of states.
 * but it is the best way to do it directly from pressures alone.
 */

/*PUBLIC*/
unsigned int UpdatePeriod(z, flagPtr)
double z[];
unsigned int *flagPtr;
{
#define INPERIOD_POS 0x8
#define PREV_POS 0x9
  unsigned int state = *flagPtr & 0x07;
  unsigned int end = *flagPtr>>4 & 0x07;
  unsigned int mask = *flagPtr & 1<<INPERIOD_POS;
  unsigned int current = (*flagPtr & 3<<PREV_POS) >> (PREV_POS-2);

  isinvalves(z);
  if (DLI && !(*flagPtr&0x2570)) *flagPtr |= 0x1000; /* Set start cycle flag */
  else *flagPtr &= 0xEFFF; /* Clear start cycle flag */
  if (DLI) *flagPtr |= 0x2000; /* Set (old) diastole flag */
  else *flagPtr &= ~0x2000;

  if (!state) return TRUE; /* No starting point seems all the time */
  current |= DLI | DLO<<1;
  *flagPtr = *flagPtr&0xFFF3 | (current&0x3)<<PREV_POS; /* Update flags */
  if (current == state)
    *flagPtr |= 1<<INPERIOD_POS; /* Set the INPERIOD flag */
  else if (current == end)
    *flagPtr &= ~(unsigned int)(1<<INPERIOD_POS); /* Clear INPERIOD flag */
  if (trial) /* Only return TRUE on up edge */
    return (!mask && (*flagPtr)>>INPERIOD_POS & 1)? TRUE : FALSE;
  else
    return (1 & *flagPtr>>INPERIOD_POS ? TRUE : FALSE);
}

/*PUBLIC*/
double getvol(z, valplace, valtype) /*only returns error (large number)
                                      if totally unreasonable */
    double z[];
    int valplace, valtype;
{
    switch (valtype) {
    case (TIME): return (zt);
    case (PRESSURE):
      switch (valplace) {
      case (SYSTEM): return(pth);
      case (PC): return (zpPv);
      case (SC): return (zpv);
      default:   return (z[valplace]);
      }
    case (VOLUME): switch (valplace) {
      case (LV): return (VlO+(zpl-pth)*zCl);
      case (RA): return (VrO+(zpa-A_TH_FRAC*pth)*Ca);
      case (SV): return (zigV+zpv*Cv);
      case (RV): return (VrO+(zpr-pth)*zCr);
      case (PA): return (VPaO+(zpPa-pth)*CPa);
      case (PV): return (VPvO+(zpPv-pth)*CPv);
      case (PC): return (0.0);
      case (SC): return (0.0);
      case (SYSTEM): return(lv);
      default: return(0.0);
      }
    case (FLOW): switch (valplace) {
      case (LV):
      case (LVI):
      case (SA): isinvalves(z); return (DLO*(zpl-zpa)/Rlo);
      case (SCI):
      case (SC): return ((zpa-zpv)/zigR);
      case (SVI):
      case (SV): isinvalves(z); return (DRI*(zpv-zpr)/Rv);
      case (RVI):
      case (RV):
      case (PA): isinvalves(z); return (DRO*(zpr-zpPa)/Rpo);
      case (PCI):
      case (PC): return ((zpPa-zpPv)/RPv);
      case (PVI):
      case (PV): isinvalves(z); return (DLI*(zpPv-zpl)/Rli);
      case (SYSTEM): return((zpa-zpv)/zigR);
      default: break;
      }
    case (CAPACITANCE): switch(valplace) {
      case (LV): return (zCl);
      case (RA): return(Ca);
      case (SC): return(0.0);
      case (SV): return(Cv);
      case (RV): return (zCr);
      case (PA): return(CPa);
      case (PC): return(0.0);
      case (PV): return(CPv);
      case (SYSTEM): break;
      default: break;
      }
    case (CAPACITANCE_SYS): switch(valplace) {
      case (LV): return(zigCl);
      case (RV): return(zigCr);
      default: return(getvol(z, valplace, CAPACITANCE));
      }
    case (CAPACITANCE_DIAS): switch(valplace) {
      case (LV): return(Clflow);
      case (RV): return(Crflow);
      case (SV): isinvalves(z); return(DRI ? zpr : zpv); /* ||| HORRIBLE HACK! Using this Capacitance_flow code for ATRIAL (Inflow) PRESSURE! */
      case (PV): isinvalves(z); return(DLI ? zpl : zpPv);
```

```
      default: return(getval(x, valplace, CAPACITANCE));
      }
    case (ZPVOLUME): switch(valplace) {
      case (LV): return(Vl0);
      case (SA): return(Va0);
      case (SC): return(0.0);
      case (SV): return(sigV);
      case (RV): return(Vr0);
      case (PA): return(VPa0);
      case (PC): return(0.0);
      case (PV): return(VPv0);
      case (SYSTEM): return(bv);
      default: return(0.0);
      }
    case (RESISTANCE): switch(valplace) {
      case (LV): return(Rlo);
      case (SA): return(0.0);
      case (SC): return(sigR);
      case (SV): return(Rv);
      case (RV): return(Rro);
      case (PA): return(0.0);
      case (PC): return(RPv);
      case (PV): return(RR);
      case (SYSTEM): break;
      default: return(0.0);
      }
    case (HEARTRATE): return(sigHR);
  }
  return((double) 0x0FFFFFFF);
}


/*PUBLIC*/
double setval(x, valplace, valtype, value) /* returns value set or 0x[illegible] */
      double x[];
      int valtype, valplace;
      double value;
{
  double SetFlowTypeValue();
  double set;
/*  if (!var_in_bounds(valtype, valplace, value)) doesn't work yet
*     return((double)0x0[illegible]);
*/

  SimFlush(); /* in Sims.c, to clear data queue of old simulated data. */
  set = SetFlowTypeValue(x, valplace, valtype, value);
  UpdateParamWindow();
  return (set);
}

static double   SetFlowTypeValue(x, valplace, valtype, value)
      double x[];
      int valtype, valplace;
      double value;
{
  double tmp;
  switch (valtype) {
  case (TIME):
  case (FLOW):
    break;
  case (PRESSURE): switch(valplace) {
      case (SYSTEM):
        xpl += value - pth;
        xpr += value - pth;
        xpPa += value - pth;
        xpPv += value - pth;
        /* ||| Pa has 1/3 transthoracic pressure on it! */
        xpa += A_TH_FRAC*(value - pth);
        pth = value;
        return(pth);
      default:
        break;
      }
      break;
    case (VOLUME): switch (valplace) {
      case (SYSTEM):
        bv = value;
        fixvolume(x);
        return(bv);
      default:
        break;
      }
      break;
    case (CAPACITANCE): switch(valplace) {
      case (SA):
        xpa *= Ca/value;
        return(Ca = value);
      case (SV):
        xpv *= Cv/value;
        return(Cv = value);
      case (PA):
        xpPa *= CPa/value;
        return(CPa = value);
      case (PV):
        xpPv *= CPv/value;
        return(CPv = value);
      case (LV):
      case (RV):
      case (SC):
      case (PC):
      case (SYSTEM):
        break;
      }
      break;
    case (CAPACITANCE_SYS): switch(valplace) {
      case (LV):
        Clsys = value; calcRefloence(); beconst();
        tmp = Cl((int) x[9]);
        xpl *= xCl/tmp; xCl = tmp; return(Clsys);
      case (RV):
        Crsys = value; calcRefloence(); beconst();
        tmp = Cr((int) x[9]);
        xpr *= xCr/tmp; xCr = tmp; return(Crsys);
      default: break;
      }
      break;
    case (CAPACITANCE_DIAS): switch(valplace) {
      case (LV):
        Cldias = value; calcRefloence(); beconst();
        tmp = Cl((int) x[9]);
        xpl *= xCl/tmp; xCl = tmp; return(Cldias);
      case (RV):
```

```
      Cdias = value; calcRefvols(); htcomp();
      tmp = Cs((int) x[P]);
      apr *= xCr/tmp; xCr = tmp; return(Cdias);
    default: break;
    }
   break;
  case (ZPVOLUME): switch(valplace) {
    case (LV):
      apl += (Vl0 - value)/xCl;
      return(Vl0 = value);
    case (RA):
      apa += (Va0 - value)/Ca;
      return(Va0 = value);
    case (SV):
      tmp = sigV; Vs0 = value; calcRefvols();
      apv += (tmp - sigV)/Cv;
      return(Vs0 = value);
    case (RV):
      apr += (Vr0 - value)/xCr;
      return(Vr0 = value);
    case (PA):
      apPa = (VPa0 - value)/CPa;
      return(VPa0= value);
    case (PV):
      apPv = (VPv0 - value)/CPv;
      return(VPv0 = value);
    case (SC):
    case (PC):
    case (SYSTEM):
      break;
    }
   break;
  case (RESISTANCE): switch(valplace) {
    case (LV): return(Rlo = value);
    case (SC):
      Ra = value;
      calcRefvols();
      return(Ra);
    case (SV): return(Rv = value);
    case (RV): return(Rro = value);
    case (PC): return(RPv = value);
    case (PV): return(Rli = value);
    case (RA):
    case (PA):
    case (SYSTEM):
      break;
    }
   break;
  case (HEARTRATE): hr = value; return(hr);
  }
  return((double) 0x0FFFFFFF); /* error: bad value or location */
 }
```

estimate.c

```
#ifndef lint
static char
*rcsid="$Header: /mit/hst/src/sim/RCS/estimate.c,v 2.1 90/08/18 18:40:12 tldavis Exp $";
#endif lint

/* R. Sah          August 1983
 * G. Moody        18 December 1984
 * T. Davis     June 1985
 * Copyright 1988 Timothy L. Davis
 * $Source: /mit/hst/src/sim/RCS/estimate.c,v $
 * Estimate initial pressures at the six nodes in the model by setting up an
 * 8x8 system of linear equations.  NOTE: uses linsys() in libm.a.
 */

#include "CVDefs.h"

extern double Tsystole(); /* from aqns.c */

estimate(p,x)
SIMULATION *p;
double x[];

{
  double a[8][8], b[8];
  double T, Tsys, Tdias;


        T = 60./p->hr;
        Tsys = Tsystole(p->hr);
        Tdias = T - Tsys;

          a[0][0] = p->Cldias;
          a[1][0] = 0.;
          a[2][0] = p->Crdias;
          a[3][0] = 0.;
          a[4][0] = p->Ca;
          a[5][0] = p->Cv;
          a[6][0] = p->CPa;
          a[7][0] = p->CPv;

          a[0][1] = p->Cldias;
          a[1][1] = -p->Clsys;
          a[2][1] = -p->Crdias;
          a[3][1] = p->Crsys;
          a[4][1] = 0.;
          a[5][1] = 0.;
          a[6][1] = 0.;
          a[7][1] = 0.;

          a[0][2] = -p->Cldias*p->Rv/Tdias;
          a[1][2] = p->Clsys*(p->Rv)/Tdias;
          a[2][2] = -1.;
          a[3][2] = 0.;
          a[4][2] = 0.;
          a[5][2] = 1.;
          a[6][2] = 0.;
          a[7][2] = 0.;

          a[0][3] = -p->Cldias*(p->Rao)/Tsys;
          a[1][3] = p->Clsys*(p->Rao)/Tsys;
          a[2][3] = 0.;
          a[3][3] = 1.;
          a[4][3] = 0.;
          a[5][3] = 0.;
          a[6][3] = -1.;
          a[7][3] = 0.;
```

```
    a[0][4] = -p->Cdias*(p->Rlo)/Tsys;
    a[1][4] = 1. + p->Csys*(p->Rlo)/Tsys;
    a[2][4] = 0.;
    a[3][4] = 0.;
    a[4][4] = -1.;
    a[5][4] = 0.;
    a[6][4] = 0.;
    a[7][4] = 0.;

    a[0][5] = -p->Cdias*(p->Ra)/T;
    a[1][5] = p->Csys*(p->Ra)/T;
    a[2][5] = 0.;
    a[3][5] = 0.;
    a[4][5] = 1.;
    a[5][5] = -1.;
    a[6][5] = 0.;
    a[7][5] = 0.;

    a[0][6] = -p->Cdias*(p->RPv)/T;
    a[1][6] = p->Csys*(p->RPv)/T;
    a[2][6] = 0.;
    a[3][6] = 0.;
    a[4][6] = 0.;
    a[5][6] = 0.;
    a[6][6] = 1.;
    a[7][6] = -1.;

    a[0][7] = -1.-p->Cdias*(p->Rli)/Tdias;
    a[1][7] = p->Csys*(p->Rli)/Tdias;
    a[2][7] = 0.;
    a[3][7] = 0.;
    a[4][7] = 0.;
    a[5][7] = 0.;
    a[6][7] = 0.;
    a[7][7] = 1.;

    b[0] = p->bv - (p->Va0+p->Vv0+p->Va0+p->VPa0+
                    p->VPv0+p->V10)
           + p->pth*
                    (p->Cldias+p->Crdias+p->CPa+p->CPv);
    b[1] = p->pth*(p->Cldias-p->Clsys-p->Crdias+p->Crsys);
    b[4] = p->pth*(p->Rlo)*(p->Csys-p->Cdias)/Tsys;
    b[3] = p->pth*(p->Rro)*(p->Csys-p->Cdias)/Tsys;
    b[5] = p->pth*(p->Ra )*(p->Csys-p->Cdias)/T;
    b[6] = p->pth*(p->RPv)*(p->Csys-p->Cdias)/T;
    b[2] = p->pth*(p->Rv )*(p->Csys-p->Cdias)/Tdias;
    b[7] = p->pth*(p->Rli)*(p->Csys-p->Cdias)/Tdias;

    lineqs(a,b);

    xpl=b[0]; /* put the estimated pressure vals in x[] using macros */
    xpr=b[2];
    xpa=b[4];
    xpv=b[3];
    xpPa=b[6];
    xpPv=b[7];
  xt  = 0.0;
  xCl = p->Cldias;
  xCr = p->Crdias;
  x[9] = 0.0;
}
```

lineqs.c

```
#ifndef lint
static char *rcsid="$Header: /mit/hst/src/sim/RCS/lineqs.c,v 2.2 90/08/28 14:45:44 tddavis Exp $";
#endif  lint
/************************************************************************/
/*                                                                      */
/*      lineqs.c         Version 1.0       Robert Saah       Aug 1983   */
/*                                   Tim Davis      Aug 1990       */
/*      $Source: /mit/hst/src/sim/RCS/lineqs.c,v $  */
/*                                                                      */
/*      routine which...                                                */
/*                                                                      */
/*      - finds solution vector to a set of N          simultaneous     */
/*        linear equations using the Gauss-Jordan reduction             */
/*        algorithm with the diagonal pivot strategy.  NxN              */
/*        matrix A contains the coefficients; B contains                */
/*        the right-hand side vector.                                   */
/*                                                                      */
/*      (Carnahan B., Luther H., Wilkes J.: Applied                     */
/*       Numerical Methods, 1969, p. 276)                               */
/*                                                                      */
/************************************************************************/

#include        <math.h>

#define  N      8
#define  EPS    0.00001

/*PUBLIC*/
int lineqs (A,B)
double  A[][N],B[];
{
 int      i,j,k;

  for (k=0; k<N; k++) {
   if (fabs(A[k][k])<EPS) {
    printf ("Error in lineqs.c\n");
    return (-1);
   }
   /* normalize the pivot row */
   for (j=k+1; j<N; j++) A[j][k]=A[j][k]/A[k][k];
   B[k]=B[k]/A[k][k];
   A[k][k]=1.;

   /* eliminate k(th) column elements except for pivot */
   for (i=0; i<N; i++) {
    if (i!=k && A[k][i]!=0.) {
         for (j=k+1; j<N; j++)
          A[j][i]-=A[k][i] * A[j][k];
         B[i]-=A[k][i] * B[k];
         A[k][i]=0.;
    }
   }
  }
  return (0);
}
```

mkdata.c

```
#ifndef lint
static char
*rcsid="$Header: /usr/hal/src/sim/RCS/mkdata.c,v 2.1 90/08/28 14:46:02 tdavis Exp $";
#endif  lint

/* Copyright 1988 Timothy L. Davis
 *$Source: /usr/hal/src/sim/RCS/mkdata.c,v $
 * Writes a block of 10 bools to the file "bdata" in the current directory.
 * To look at this binary data directly, use the companion program "readata".
 */

#include <fcntl.h>
#include <stdio.h>
#include <signal.h>
#include "CVDefs.h"

extern SIMULATION *getsimulation();
extern double gauss();

void SimDoneBug(n)
double x[];
{
 printf("V1:   %lg  P1:   %lg  C1:   %lg  t:%lg i:%d Va:   %lg\n",
        gauss(x, LV, VOLUME), x[8], x[7], x[6], (int)x[9],
        gauss(x,SA,VOLUME));
}

int fpe_in_simulate()
{ panic("FPE!");
}
main()
{
SIMULATION *parameters;
extern modparam_handle();
extern double xmin, xmax, ymin, ymax;
int i;
double x[10];

parameters = getsimulation(NORMAL);
initoparam(parameters);

estimate(parameters, x);
beeper();

while (x[6] < 10) simulate(x, 100);

printf("simulate:   terminated!\n");
fflush(stdout);

}

PostScriptPlotting(c)
    char *c;
{  char cc[256];
  sprintf(cc, "awrite -q $USER -m %s",c);
  system(cc);
}

SetParamFromSlider(p, t, v, fd)
int p, t, fd;
double v;
{
 ;
}

void UpdateParamWindow()
{
 ;
}
```

mktbl.c

```
#ifndef lint
static char *rcsid="$Header: /usr/hal/src/sim/RCS/mktbl.c,v 2.1 90/08/18 18:40:26 tdavis Exp $";
#endif  lint
/*
 *       $Source: /usr/hal/src/sim/RCS/mktbl.c,v $
 *
 * This program generates capacitance file and dc/dt file based on model
 * first suggested by Bob Sah in his original notes on the simulator.  This
 * fixes a bug which occurred when the original elastance minimum was 0.
 * This version has minimum elastance of 0.1, maximum of 1, as in paper
 * in CRC Biomedical Engineering  ( I'll find the reference later.)
 * Relaxation occurs in about 1/2 the time as contraction, both similar
 * to sinusoidal elastance.
 * Careful to round to nearest integer!  Using floor(x+ 0.5) to do this.
 * dcap is hand-done derivative of cap with respect to i.
 */

#include <stdio.h>
#include <math.h>
#include "CVDefs.h"

main()
{
 register double elast;
 register double cap, dcap;
 register int i;
 double pi = 3.14159265;
 FILE *tblo, *tbldodt;

 tblo = fopen("tblo", "w+");  tbldodt = fopen("tbldodt", "w+");
 if (!tblo || !tbldodt) {
  printf("Couldn't open file to write!\n");
  exit(1);
 }
 fprintf(tblo,  "%d, \n\n\n", 99000);    /* these are the max value for c */
 fprintf(tbldodt, "%d, \n\n\n", 99000);

 for (i=0; i < SYSTOLEBEGS; i++) { /* during slow contraction */
  elast = .1 + .9*(cos((i - SYSTOLEBEGS)*pi/SYSTOLEBEGS) + 1)/2;
  cap = 10000.0*(1.0/elast - 1.0);
  fprintf(tblo,  "%d, \n", (int) floor(cap + 0.5));

  dcap = 1000*(.9*(pi/SYSTOLEBEGS)*0.45 *
    sin((i-SYSTOLEBEGS)*pi/SYSTOLEBEGS) / (elast*elast);
  fprintf(tbldodt,  "%d,  \n", (int) floor(dcap + 0.5));
 }

 for (i=SYSTOLEBEGS; i < TBLSIZEPLUS2; i++) { /* during fast relaxation */
```

```
    elast = .1 + .9*(cos((i - SYSTOLESBOS)*pi*2/SYSTOLESBOS) + 1)/2;
    cap = 10000.0*(1.0/elast - 1.0);
    fprintf(fdds, "%d, \n", (int) floor(cap + 0.5));

    dcap = 10000.0*(2*pi/SYSTOLESBOS)*0.45 *
      sin((i-SYSTOLESBOS)*2*pi/SYSTOLESBOS) / (elast*elast);
    fprintf(fddsdt, "%d, \n", (int) floor(dcap + 0.5));
  }
}
```

reflex.c

```
/*
 * Baroreflex extensions to sqna.c in the Cardiovascular Simulator.
 * Timothy Davis, 7/30/89
 * Based loosely on the DeBoer heat-to-heat model for
 * baroreceptor control of heart rate, contractility,
 * and peripheral resistance, with added control of venous blood volume.
 *
 * $Source: /var/tldavis/src/sim/BCS/reflex.c,v $
 * $Log:            reflex.c,v $
 * Revision 2.5  91/02/04  16:43:51  tldavis
 * Updated arterial pressure setpoint to 98 mm Hg.
 *
 * Revision 2.4  90/08/28  14:46:31  tldavis
 * Tuned.
 *
 * Revision 2.3  90/08/18  18:40:31  tldavis
 * New location of CVDefs.
 * also bugfixes to X interface sections, ref. to Interface.c for
 * dialog boxes.
 *
 * Revision 2.2  90/08/10  10:58:08  dst
 * moved Xaw header files to X11/Xaw for R4
 *
 * Revision 2.1  90/08/07  16:07:26  tldavis
 * Removed unused variables.
 *
 * Revision 2.0  90/08/05  18:02:54  tldavis
 * Final X11R3 version.
 *
 * Revision 1.6  90/05/31  16:27:06  tldavis
 * Pre-ACES checkin.
 *
 */
#include "CVDefs.h"
#include <math.h>

#ifndef NO_X_WINDOWS
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Shell.h>
#include <X11/Xaw/Dialog.h>
#include <X11/Xaw/Command.h>
#include <X11/Xaw/Form.h>
#include "../nl/NumberLine.h"
#include "../main/Interface.h" /* DialogCreate(),PlaceDialog(),XtRDialog() */

/*PUBLIC*/ Boolean reflexesOn = 0;

#else NO_X_WINDOWS
/*PUBLIC*/ void SimFlush()
{
; /*NOOP*/
}
typedef char Boolean;
#define True 1
#define False 0

/*PUBLIC*/ Boolean reflexesOn = 1;
#endif NO_X_WINDOWS

/* The following "sigma" variables are used in sqna.c */
/*PUBLIC*/
double sigHR = NHR, sigR = NRA, sigV = NVV0, sigCl = NCLSYS, sigCr = NCRSYS;
extern double Ir, Ra, Vv0, Clsys, Crsys, Cc; /* from sqna.c */

#define SETPOINT 98 /* Set as close as possible to maintaining homeostasis. */
static Boolean hrActive = True, cActive = True, rActive = True, vActive = True;
static double alphaGain = 9, betaGain = 9, paraGain = 9, aSetpoint = SETPOINT;
static double alphaTone = 0, betaTone = 0, paraTone = 0; /* baseline */

/*************************************************************************
 * Signal history manipulation routines:
 * save the last 30 seconds in 40 ms increments for
 * computing time-delayed nervous response signals.
 *************************************************************************/

#define Sgran (0.5) /* 500 millisecond granularity in signal history */
#define Slength (60) /* for 60 * 0.5 = 30 seconds of data. */

static double Shist[Slength]; /* queue representing arterial b.p. */
static int top=0; /* head of the queue */

/* Enqueue a blood pressure value */
static void saveBP(signal)
    double signal;
{
 top = (top + Slength - 1) % Slength; /* wraparound modulus arithmetic */
 Shist[top] = signal;
}

static void scaleSignal(scale)
double scale;
{
 register int i = 0;
 for (i = 0; i < Slength; i++)
  Shist[i] *= scale;
}

#ifdef NOTDEF
static void translateSignal(shift)
double shift;
{
 register int i = 0;
 double max = 15*sin(100000.0);
 double min = 15*sin(-100000.0);

 for (i = 0; i < Slength; i++) {
  Shist[i] += shift;
```

```
    if (Shist[i] > max) Shist[i] = max;
    if (Shist[i] < min) Shist[i] = min;
  }
}
#endif NOTDEF

/* Compute weighted sum of past pressures */
static double sumResponse(coeffs, start, end)
    double coeffs[]; /*Coefficients are in reverse (most recent 1st) order*/
    int start, end;/* to optimize in case of short response characteristics */
{
 double sum = 0.0;  int i;
 for (i = start; i < end; i++)
   sum += coeffs[i] * Shist[(Sp + i) % Slength]; /* more modulus math */
 return sum;
}

/* Prepare a list of coeffs to match given triangle ramp parameters. */
static void mkRamp(coeffs, start, peak, end, area)
    double *coeffs; /* pointer to coeffs buffer*/
    double area;    /* total sumResponse  for unity b.p. signal */
    double start, peak, end; /* ramp timing in seconds(to convert to index)*/
{
 int i;
 double sum = 0.0;
 start /= Sgran; peak /= Sgran; end /= Sgran; /* Convert to index */

 for (i = start; i < peak; i++)
   sum += coeffs[i] = (i-((int)start))/(peak-start);
 for (i = peak; i < end; i++)
   sum += coeffs[i] = (((int)end) - i)/(end - peak);
 for (i = start; i < end; i++)
   coeffs[i] *= area/sum; /* normalizing to required total sum
                             (analogous to area in continuous time) */
}

/*****************************************************************************
 * aCoeff is to be a list of timing coefficients for alpha-sympathetic control.
 * bCoeff is coefficients for beta-sympathetic control.
 * pCoeff contains parasympathetic timing information.
 *****************************************************************************/

static double aCoeff[Slength], bCoeff[Slength], pCoeff[Slength];
static int astart, aend, bstart, bend, pstart, pend;

/*PUBLIC*/ /* used in Event.c */
void setupCoeffs() /* Here we define the shapes of coefficient arrays */
{
 resetSignal(0.0); /*zero out everything*/
 mkRamp(aCoeff, 2.0, 5.0, 30.0, 1.0);
 astart = 2.0 / Sgran; aend = 30.0 / Sgran;
 mkRamp(bCoeff, 2.0, 5.0, 30.0, 1.0);
 bstart = 2.0 / Sgran; bend = 30.0 / Sgran;
 mkRamp(pCoeff, 0.0, 0.5, 1.0, 1.0);
 pstart = 0.0 / Sgran; pend = 1.0 / Sgran;
}

/*****************************************************************************
 * reflex arc definitions:
 *
 *****************************************************************************/

/* Reflex arc replacement when reflexes are off */
static void NoReflex()
{
 sigHR = hr;
 sigCl = Clsys;
 sigCr = Crsys;
 sigR  = Rs;
 sigV  = Vv0;
}

/* Reflex arc calculations, performed at 2 times per beat granularity */
static void Reflexes()
{
 double i, r, v, p; /* DeBoer's lower-case deviation variables */
 double alphaResp, betaResp, paraResp, I;
 static double dhr = 0.2   / 0.5083; /* Max Cl deviation / Max i */
 static double ms  = 750.0 / 0.5083; /* Max V0 deviation / Max i */

 /* Calculate current levels of autonomic reflex activity based on past ABP*/
 alphaResp = alphaGain * sumResponse(aCoeff, astart, aend) * 0.001;
 betaResp  = betaGain  * sumResponse(bCoeff, bstart, bend) * 0.001;
 paraResp  = paraGain  * sumResponse(pCoeff, pstart, pend) * 0.001;

 /* Instrbeat interval (small-signal model) from DeBoer */
 i = (baActive ? betaResp + paraResp : 0.0) - betaTone + paraTone;
    /* HR determined by both symp. and parasymp. responses */

 /* Systolic capacitance linear feedback, scaled to change Clsys and Crsys/3
  * by a maximum of 0.2 sunits/ml.  This conforms to Guyton's assertion that
  * contractility can double with exercise, and stays within simulation bound
  * of 0.2 for Clsys.
  */
 p = dhr * ((aActive ? betaResp : 0.0) - betaTone)^2;
    /* Clsys and Crsys/3 deviations */

 /* Peripheral resistance (small-signal model) from DeBoer */
 r = - ((aActive ? alphaResp : 0.0) - alphaTone)^2 / Cs;

 /* Venous zero-pressure volume (reflecting venous tone) feedback scaled for
  * a maximum of 500cc deviation from reference.  This comes from Guyton,
  * who says venous pooling can vary between 400cc and 2600cc from tense.
  */
 v = ms * ((vActive ? alphaResp : 0.0) - alphaTone)^2;

 /* Add either feedback signal deviation or pure tonus data
  * to controlled variables.
  */
 I = 60./hr + i;
 sigHR = (I < 0.25) ? 60/0.25 : 60./I;   /* max HR signal of 240 */
 if (sigHR > 250) sigHR = 250.0;
 if (sigHR < 0) sigHR = 0.0;
 sigCl = Clsys + p;  if (sigCl < 0.2) sigCl = 0.2;
 sigCr = Crsys + 3*p; if (sigCr < 0.6) sigCr = 0.2;
 sigR  = Rs + r;     if (sigR  < 0.1) sigR  = 0.1;
 sigV  = Vv0+ v;     if (sigV  < 0.0) sigV  = 0.0;
}

/* The following simply updates the baroreflex on demand for the benefit of
 * correct() in aqua.c which needs an up-to-date system.
 */
```

```
/*PUBLIC*/ void calcReflexes()
{
  if (reflexesOn)
    Reflexes();
  else
    NoReflex();
}

/* SANODE tickles the s-a node to determine if a new beat should occur. It
 * does this by integrating a heart rate signal until
 * average_HR*This_interval >= 1.0.
 * It also tickles the baroreflex every 40 milliseconds.
 */
/*PUBLIC*/
void sanode(x, dt)
     double x[];
     double dt;
{
  static double cum_hr = 0.0, cum_t = 0.0;
  static double cum_t2 , cum_abp = 0.0; /* for 40ms abp average */
  static int abp_n = 0, n = 0;

  cum_t2 += dt;
  abp_n++;
  cum_abp += x[1];
  if (cum_t2 > Sgma) {
    /* save signal for the last 40 milliseconds */
    double s; /* instantaneous baroreceptor (ascending nerve) signal */
    s = 18*atan((cum_abp/abp_n - altsetpoint)/18);  /* from DeBoer */
    saveR(s); /* enqueue this average pressure */
    cum_t2 -= Sgma; abp_n = 0; cum_abp = 0; /*start next interval avging*/
    if (reflexesOn)  Reflexes();
    else             NoReflex();
  }

  cum_hr += sigHR;
  cum_t  += dt;
  n++;
  /*
   * Wait until area under the hr signal vs. time curve reaches 1.00,
   * then start the next beat, as long as we have passed the absolute
   * refractory period of 1/5 of a beat length since systole.
   * Setting cum_t to -0.1 gives a 0.1 sec P-R conduction delay.
   */
 if ((cum_t*cum_hr/n >= 60.0) && (x[9] > TBLSECS * 1.2)) {
   /* start a new beat */
   x[9] = 0.0; /* reset this index, which starts a beat. */
   cum_hr = 0.0; cum_t = 0.0; n = 0; /* reset summation "IPFM" integral */
   beatreset(); /* reset simulation constants for next beat */

#ifdef DEBUG
    if (reflexesOn)
     printf("signal:%4d, hr:%6.3d, R:%6.3f, Cl:%6.3f, Cr:%6.3f, Vv0:%6.0f\n",
            (int) Shist[top], (int) sigHR, (float) sigR, (float) sigCl,
            (float) sigCr, (float) sigV);
#endif DEBUG
  }
}

#ifndef NO_X_WINDOWS
/**********************************************************************
 *
 * User interface to Baroreflex feedback system.
 *
 **********************************************************************/

static Widget hrButton, cButton, dButton, vButton, modButton,
  reflexActiveButton;
extern Display *dpy;
extern Window root;
extern int screen;

extern int Stopped; /* from Event.c */
extern void StopSimulator(), ResumeSimulator();

Boolean Paused = False;

/*ARGSUSED*/
static void ClearSome(w, client, call)
Widget w;
caddr_t client, call;
{
  int *some_ptr = (int *) client;
  (*some_ptr)++;
  if (Paused) ResumeSimulator();
  Paused = False;
}

/*ARGSUSED*/
static void UseDataCallback(w, client, call)
Widget w;
caddr_t client, call;
{
  Arg args[2];
  XEvent ev;

  while (XCheckWindowEvent(dpy, XtWindow(w), ButtonPressMask, &ev)) ;

  XtSetArg(args[0],XtNlabel,reflexesOn?"Reflexes are ON":"Reflexes are OFF");
  XtSetValues(reflexActiveButton, args, 1);
  if (reflexesOn)   XtSetArg(args[0], XtNsensitive, True);
  else              XtSetArg(args[0], XtNsensitive, False);

  XtSetValues(hrButton, args, 1);
  XtSetValues(cButton, args, 1);
  XtSetValues(dButton, args, 1);
  XtSetValues(vButton, args, 1);
  XtSetValues(modButton, args, 1);
}

/*ARGSUSED*/
static void ResetCallback(w, client, call)
Widget w;
caddr_t client, call;
{
  resetSignal(0.0); /* zero-out past signal */
  UseDataCallback(w, client, call);
}

static void getReflexToggle(w, client, call)
    Widget w;
    caddr_t client, call;
```

```
{
  static int modal_semn = 1;
  XEvent ev;

  if (!client) {
    reflexesOn = !reflexesOn;
    if (reflexesOn) {
      /* Determine if we want to start from rest state or use current data. */
      if (modal_semn) {
          Widget dialog, rest, usedata;

          modal_semn--;
          Paused = !Stopped;
          if (Paused) StopSimulator();

          dialog = DialogCreate(w, "Choose initial reflex response", (char *)0);

          usedata = XtCreateManagedWidget("Use current data",commandWidgetClass,
                                          dialog,(Arg *)NULL, 0);
          XtAddCallback(usedata, XtNcallback, UseDataCallback, (caddr_t) 0);
          XtAddCallback(usedata, XtNcallback, ClearSema, (caddr_t) &modal_semn);
          XtAddCallback(usedata, XtNcallback, KillDialog, (caddr_t) dialog);
          rest = XtCreateManagedWidget("Reset to rest conditions",
                                       commandWidgetClass,
                                       dialog, (Arg *)NULL, 0);
          XtAddCallback(rest, XtNcallback, RestCallback, (caddr_t) 0);
          XtAddCallback(rest, XtNcallback, ClearSema, (caddr_t) &modal_semn);
          XtAddCallback(rest, XtNcallback, KillDialog, (caddr_t)dialog);
          PlaceDialog(dialog, w);
      }
      while (XCheckWindowEvent(dpy, XtWindow(w), ButtonPressMask, &ev)) ;
    } else UseDataCallback(w, client, call);
  } else UseDataCallback(w, client, call);
}

static void getLimbToggle(w, client, call)
    Widget w;
    caddr_t client, call;
{
  Arg args[3];
  Boolean *limb = (Boolean *) client;
  if (!call) *limb = ! *limb;
  XtSetArg(args[0],XtNlabel,*limb?"ACTIVE":"inactive");
  XtSetValues(w, args, 1);
}

/*ARGSUSED*/
static void sympCallback(w, client, call)
    Widget w;
    caddr_t client, call;
{
  double present, *presentPtr = (double *) client;
  int done;
  NumberLineGetValue(w, &present, &done);
#ifdef DEBUG
  printf("Symp.    Callback:    present = %lf\n", present);
#endif
  *presentPtr = present;
}
```

```
#ifdef NOTDEF
/* THIS ISN'T CURRENTLY USED:  nonphysiologic all-time setpoint change.
/*
 * setpointChange() callback implements a nonphysiologic setpoint change for
 * all time, changing the history of the baroreceptor to
 * pretend the setpoint was this all along.
 * I consider this nonphysiologic and thus unintuitive.
 */
/*ARGSUSED*/
static void setpointChange(w, client, call)
    Widget w;
    caddr_t client, call;
{
  static double previous = SETPOINT;
  double present;
  int done;
  NumberLineGetValue(w, &present, &done);
  /* Change the all signal from the baroreceptor to center on the
    NEW setpoint */
  translateSignal(previous - present);
  previous = present;
}
#endif NOTDEF

/*ARGSUSED*/
static void destroyNotify(w, client, call)
    Widget w;
    caddr_t client, call;
{
  *((Boolean *)client) = False;
}

/*ARGSUSED*/
static void dummyHandler(w, client, event)
    Widget w; caddr_t client; XEvent *event;
{ ; /*nothing*/
}

/*ARGSUSED*/
static void modReflexCallback(w, client, call)
    Widget w;
    caddr_t client, call;
{
  Arg args[20];
  int i, j;
  static Widget shell, form, titleLabel, alphaNl, betaNl, paraNl, setpointNl,
                stNl, btNl, ptNl;
  static double amin = 0, pmin = 0, amax = 20, pmax = 20, adef = 9, pdef = 9;
  static double tmin = -1, tmax = 1, tdef = 0;
  static double smin = 0, smax = 200, sdef = 90;
  static Boolean Exists = False;

  if (Exists) return;
  else Exists = True;
  i=0;
  XtSetArg(args[i], XtNallowShellResize, True);i++;
  XtSetArg(args[i], XtNiconName, "Baroreceptor Reflex Parameters");i++;
#ifdef XAW_BC
  shell = XtCreateApplicationShell("baroreceptor_parameters",
                                   topLevelShellWidgetClass, args, i);
#else ! XAW_BC
```

```
    shell = XtAppCreateShell(NULL,"CVsim", topLevelShellWidgetClass,dpy,args,i);
#endif XAW BC
    XtAddCallback(shell, XtNdestroyCallback, destroyNotify, (caddr_t)&Exists);
    XtAddEventHandler(shell, ButtonPressMask | ButtonReleaseMask, False,
                      dummyHandler, (caddr_t)0);
    form = XtCreateManagedWidget("Baroreceptor Reflex", formWidgetClass,
                                 shell, (Arg *)0, 0);
    i=0;
    XtSetArg(args[i], XtNvertDistance, 4);i++;
    XtSetArg(args[i], XtNresizable, True);i++;
    j = i;
    XtSetArg(args[i], XtNborderWidth, 0);i++;
    titleLabel = XtCreateManagedWidget("BARORECEPTOR REFLEX PARAMETERS",
                                       labelWidgetClass, form, args, i);
    XtSetArg(args[j], XtNwidth, 200);j++;
    i=j;
    XtSetArg(args[i], XtNfromVert, titleLabel);i++;
    XtSetArg(args[i], XtNlabel, "ABP Setpoint  (mmHg) ");i++;
    XtSetArg(args[i], XtNminimum, &smin);i++;
    XtSetArg(args[i], XtNmaximum, &smax);i++;
    XtSetArg(args[i], XtNpresent, &aSetpoint);i++;
    XtSetArg(args[i], XtNdefault, &sdef);i++;
    XtSetArg(args[i], XtNprevious, &aSetpoint);i++;
    setpointNl = XtCreateManagedWidget("numberLine", numberLineWidgetClass,
                                       form, args, i);
    XtAddCallback(setpointNl, XtNcallback, sympCallback,(caddr_t) &aSetpoint);
#ifdef NOTDEF
/* Nonphysiologic "total" setpoint changes. */
    XtAddCallback(setpointNl, XtNcallback, setpointChange, (caddr_t) &aSetpoint);
#endif NOTDEF
    /* Left side */
    i=j;
    XtSetArg(args[i], XtNfromVert, setpointNl);i++;
    XtSetArg(args[i], XtNlabel, "Periph.    Sympathetic Tone");i++;
    XtSetArg(args[i], XtNminimum, &tmin);i++;
    XtSetArg(args[i], XtNmaximum, &tmax);i++;
    XtSetArg(args[i], XtNpresent, &alphaTone);i++;
    XtSetArg(args[i], XtNdefault, &tdef);i++;
    XtSetArg(args[i], XtNprevious, &alphaTone);i++;
    atNl = XtCreateManagedWidget("numberLine", numberLineWidgetClass,
                                 form, args, i);
    XtAddCallback(atNl, XtNcallback, sympCallback,(caddr_t) &alphaTone);
    i=j;
    XtSetArg(args[i], XtNfromVert, atNl);i++;
    XtSetArg(args[i], XtNlabel, "Heart Sympathetic Tone");i++;
    XtSetArg(args[i], XtNminimum, &tmin);i++;
    XtSetArg(args[i], XtNmaximum, &tmax);i++;
    XtSetArg(args[i], XtNpresent, &betaTone);i++;
    XtSetArg(args[i], XtNdefault, &tdef);i++;
    XtSetArg(args[i], XtNprevious, &betaTone);i++;
    btNl = XtCreateManagedWidget("numberLine", numberLineWidgetClass,
                                 form, args, i);
    XtAddCallback(btNl, XtNcallback, sympCallback,(caddr_t) &betaTone);
    i=j;
    XtSetArg(args[i], XtNfromVert, btNl);i++;
    XtSetArg(args[i], XtNlabel, "Parasympathetic Tone");i++;
    XtSetArg(args[i], XtNminimum, &tmin);i++;
    XtSetArg(args[i], XtNmaximum, &tmax);i++;
    XtSetArg(args[i], XtNpresent, &paraTone);i++;
    XtSetArg(args[i], XtNdefault, &tdef);i++;
    XtSetArg(args[i], XtNprevious, &paraTone);i++;
    ptNl = XtCreateManagedWidget("numberLine", numberLineWidgetClass,
                                 form, args, i);
    XtAddCallback(ptNl, XtNcallback, sympCallback,(caddr_t) &paraTone);
    /* right side */
    i=j;
    XtSetArg(args[i], XtNfromVert, setpointNl);i++;
    XtSetArg(args[i], XtNfromHoriz, setpointNl);i++;
    XtSetArg(args[i], XtNlabel, "Periph.    Sympathetic Gain");i++;
    XtSetArg(args[i], XtNminimum, &gmin);i++;
    XtSetArg(args[i], XtNmaximum, &gmax);i++;
    XtSetArg(args[i], XtNpresent, &alphaGain);i++;
    XtSetArg(args[i], XtNdefault, &gdef);i++;
    XtSetArg(args[i], XtNprevious, &alphaGain);i++;
    alphaNl = XtCreateManagedWidget("numberLine", numberLineWidgetClass,
                                    form, args, i);
    XtAddCallback(alphaNl, XtNcallback, sympCallback,(caddr_t) &alphaGain);
    i=j;
    XtSetArg(args[i], XtNfromVert, alphaNl);i++;
    XtSetArg(args[i], XtNfromHoriz, setpointNl);i++;
    XtSetArg(args[i], XtNlabel, "Heart Sympathetic Gain");i++;
    XtSetArg(args[i], XtNminimum, &gmin);i++;
    XtSetArg(args[i], XtNmaximum, &gmax);i++;
    XtSetArg(args[i], XtNpresent, &betaGain);i++;
    XtSetArg(args[i], XtNdefault, &gdef);i++;
    XtSetArg(args[i], XtNprevious, &betaGain);i++;
    betaNl = XtCreateManagedWidget("numberLine", numberLineWidgetClass,
                                   form, args, i);
    XtAddCallback(betaNl, XtNcallback, sympCallback,(caddr_t) &betaGain);
    i=j;
    XtSetArg(args[i], XtNfromVert, betaNl);i++;
    XtSetArg(args[i], XtNlabel, "Parasympathetic Gain");i++;
    XtSetArg(args[i], XtNfromHoriz, setpointNl);i++;
    XtSetArg(args[i], XtNminimum, &gmin);i++;
    XtSetArg(args[i], XtNmaximum, &gmax);i++;
    XtSetArg(args[i], XtNpresent, &paraGain);i++;
    XtSetArg(args[i], XtNdefault, &gdef);i++;
    XtSetArg(args[i], XtNprevious, &paraGain);i++;
    paraNl = XtCreateManagedWidget("numberLine", numberLineWidgetClass,
                                   form, args, i);
    XtAddCallback(paraNl, XtNcallback, sympCallback,(caddr_t) &paraGain);

    XtRealizeWidget(shell);
    XtPopup(shell, XtGrabNone);
    XtMapWidget(shell);
}

/*PUBLIC*/
void ReflexesOff()
{
  if (reflexActiveButton)
    getReflexToggle((Widget)reflexActiveButton,(caddr_t)reflexesOn, (caddr_t)0);
}

/*ARGSUSED*/
/*PUBLIC*/
void ReflexButton(widget, client, call)
Widget widget;
caddr_t client, call;
{
  static Widget titleLabel, hrLabel, cLabel, rLabel, vLabel,
                form, shell;
```

```
  Arg args[20];
  int i, j;
  static Boolean Exists = False;

  if (Exists) return;
  else Exists = True;

  i=0;
  XtSetArg(args[i], XtNallowShellResize, True);i++;
  XtSetArg(args[i], XtNiconName, "Baroreceptor Reflex Control");i++;
#ifdef XAW_BC
  shell = XtCreateApplicationShell("baroreceptor_control"
                                   topLevelShellWidgetClass, args, i);
#else ! XAW_BC
  shell = XtAppCreateShell(NULL,"CVsim",topLevelShellWidgetClass,dpy,args,i);
#endif XAW_BC
  XtAddCallback(shell, XtNdestroyCallback, destroyNotify, (caddrt)&Exists);
  XtAddEventHandler(shell, ButtonPressMask | ButtonReleaseMask, False,
                    dummyHandler, (caddrt)0);
  form = XtCreateManagedWidget("Baroreceptor Reflex", formWidgetClass,
                               shell, (Arg *)0, 0);
  i=0;
  XtSetArg(args[i], XtNtop, XtChainTop);i++;
  XtSetArg(args[i], XtNbottom, XtChainTop);i++;
  XtSetArg(args[i], XtNleft, XtChainLeft);i++;
  XtSetArg(args[i], XtNright, XtChainLeft);i++;
  XtSetArg(args[i], XtNvertDistance, 8);i++;
  XtSetArg(args[i], XtNresizable, True);i++;
  j = i;
  XtSetArg(args[i], XtNborderWidth, 0);i++;
  titleLabel = XtCreateManagedWidget("BARORECEPTOR REFLEX CONTROL",
                                     labelWidgetClass, form, args, i);
  i=j;
  XtSetArg(args[i], XtNfromHoriz, titleLabel); i++;
  XtSetArg(args[i], XtNborderWidth, 3); i++;
  reflexActiveButton = XtCreateManagedWidget("ReflexesActiveButton",
                                     commandWidgetClass, form, args, i);
  XtAddCallback(reflexActiveButton,XtNcallback, getReflexToggle, (caddrt)0);

  i=j;
  XtSetArg(args[i], XtNfromVert, titleLabel);i++;
  XtSetArg(args[i], XtNborderWidth, 0);i++;
  hrLabel = XtCreateManagedWidget("Heart Rate feedback:", labelWidgetClass,
                                  form, args, i);
  i=j;
  XtSetArg(args[i], XtNfromVert, titleLabel);i++;
  XtSetArg(args[i], XtNfromHoriz, titleLabel);i++;
  XtSetArg(args[i], XtNborderWidth, 3);
  hrButton = XtCreateManagedWidget("hrButton", commandWidgetClass,
                                   form, args, i);
  XtAddCallback(hrButton, XtNcallback, getLimbToggle, (caddrt) &hrActive);
  i=j;
  XtSetArg(args[i], XtNfromVert, hrLabel);i++;
  XtSetArg(args[i], XtNborderWidth, 0);i++;
  cLabel = XtCreateManagedWidget("Contractility feedback:", labelWidgetClass,
                                 form, args, i);
  i=j;
  XtSetArg(args[i], XtNfromVert, hrLabel);i++;
  XtSetArg(args[i], XtNfromHoriz, titleLabel);i++;
  XtSetArg(args[i], XtNborderWidth, 3);
```

```
  cButton = XtCreateManagedWidget("cButton",commandWidgetClass,form,args, i);
  XtAddCallback(cButton, XtNcallback, getLimbToggle, (caddrt) &cActive);
  i=j;
  XtSetArg(args[i], XtNfromVert, cLabel);i++;
  XtSetArg(args[i], XtNborderWidth, 0);i++;
  rLabel=XtCreateManagedWidget("Art.    Resistance feedback:",labelWidgetClass,
                               form, args, i);
  i=j;
  XtSetArg(args[i], XtNfromVert, cLabel);i++;
  XtSetArg(args[i], XtNfromHoriz, rLabel);i++;
  XtSetArg(args[i], XtNborderWidth, 3);
  rButton = XtCreateManagedWidget("rButton",commandWidgetClass,form,args,i);
  XtAddCallback(rButton, XtNcallback, getLimbToggle, (caddrt) &rActive);
  i=j;
  XtSetArg(args[i], XtNfromVert, rLabel);i++;
  XtSetArg(args[i], XtNborderWidth, 0);i++;
  vLabel = XtCreateManagedWidget("Venous tone feedback:", labelWidgetClass,
                                 form, args, i);
  i=j;
  XtSetArg(args[i], XtNfromVert, rLabel);i++;
  XtSetArg(args[i], XtNfromHoriz, rLabel);i++;
  XtSetArg(args[i], XtNborderWidth, 3);
  vButton = XtCreateManagedWidget("vButton",commandWidgetClass,form,args,i);
  XtAddCallback(vButton, XtNcallback, getLimbToggle, (caddrt) &vActive);

  i=j;
  XtSetArg(args[i], XtNfromVert, vLabel);i++;
  XtSetArg(args[i], XtNborderWidth, 3);
  modButton = XtCreateManagedWidget("Modify Reflex Parameters",
                                    commandWidgetClass, form, args, i);
  XtAddCallback(modButton, XtNcallback, modReflexCallback, (caddrt)0);

  XtRealizeWidget(shell);
  XtPopup(shell, XtGrabNone);
  XtMainLoop(shell);

  getReflexToggle((Widget)reflexActiveButton, (caddrt)1, (caddrt)0);
  getLimbToggle(hrButton, (caddrt)&hrActive, (caddrt)1);
  getLimbToggle(cButton, (caddrt)&cActive, (caddrt)1);
  getLimbToggle(rButton, (caddrt)&rActive, (caddrt)1);
  getLimbToggle(vButton, (caddrt)&vActive, (caddrt)1);
}

#endif NO_X_WINDOWS
/*** end of Reflex Interface ***********************************************/
```

rk4.c

```
void rk4(y,dydx,x,h,yout,derivs)
double y[],dydx[],x,h,yout[];
void (*derivs)();          /* ANSI: void (*derivs)(double,double *,double *); */
{
        double hh,h6,dym[6],dyt[6],yt[10];
        int xh;
        extern double Cinc();

        hh=h*0.5;
        h6=h/6.0;
        xh=x+Cinc(hh);
        yt[6]=y[6]; yt[7]=y[7]; yt[8]=y[8]; yt[9]=y[9];
```

```
        yt[0]=y[0]+hh*dydx[0];
        yt[1]=y[1]+hh*dydx[1];
        yt[2]=y[2]+hh*dydx[2];
        yt[3]=y[3]+hh*dydx[3];
        yt[4]=y[4]+hh*dydx[4];
        yt[5]=y[5]+hh*dydx[5];
        (*derivs)(dyt, xh,yt);

        yt[0]=y[0]+hh*dyt[0];
        yt[1]=y[1]+hh*dyt[1];
        yt[2]=y[2]+hh*dyt[2];
        yt[3]=y[3]+hh*dyt[3];
        yt[4]=y[4]+hh*dyt[4];
        yt[5]=y[5]+hh*dyt[5];
        (*derivs)(dym, xh,yt);

        yt[0]=y[0]+h*dym[0];  dym[0] += dyt[0];
        yt[1]=y[1]+h*dym[1];  dym[1] += dyt[1];
        yt[2]=y[2]+h*dym[2];  dym[2] += dyt[2];
        yt[3]=y[3]+h*dym[3];  dym[3] += dyt[3];
        yt[4]=y[4]+h*dym[4];  dym[4] += dyt[4];
        yt[5]=y[5]+h*dym[5];  dym[5] += dyt[5];
        (*derivs)(dyt, (int)(x+Cinc(h)),yt);

        yout[0]=y[0]+h6*(dydx[0]+dyt[0]+2.0*dym[0]);
        yout[1]=y[1]+h6*(dydx[1]+dyt[1]+2.0*dym[1]);
        yout[2]=y[2]+h6*(dydx[2]+dyt[2]+2.0*dym[2]);
        yout[3]=y[3]+h6*(dydx[3]+dyt[3]+2.0*dym[3]);
        yout[4]=y[4]+h6*(dydx[4]+dyt[4]+2.0*dym[4]);
        yout[5]=y[5]+h6*(dydx[5]+dyt[5]+2.0*dym[5]);
}
```

rkqc.c

```
#include <math.h>

#define PGROW -0.29
#define PSHRINK -0.25
#define FCOR 0.06666666          /* 1.0/15.0 */
#define SAFETY 0.9
#define ERRCON 6.0e-4

int rkqc(y,dydx,x,htry,eps,yscal,hdid,hnext,derivs)
    double y[],dydx[], *x,htry,eps,yscal[],*hdid,*hnext;
    void (*derivs)();          /* ANSI: void (*derivs)(double,double *,double *); */
{
  double xsav,hh,h,temp,errmax;
  double dysav[6],ysav[10],ytemp[10];
  extern void rk4();
  extern double Cinc();

  xsav=(*x);

  ysav[0]=y[0];
  ysav[1]=y[1];
  ysav[2]=y[2];
  ysav[3]=y[3];
  ysav[4]=y[4];
  ysav[5]=y[5];
  dysav[0]=dydx[0];
  dysav[1]=dydx[1];
  dysav[2]=dydx[2];
  dysav[3]=dydx[3];
  dysav[4]=dydx[4];
  dysav[5]=dydx[5];

  h=htry;
  ytemp[6]=ysav[6]=y[6];  ytemp[7]=ysav[7]=y[7];
  ytemp[8]=ysav[8]=y[8];  ytemp[9]=ysav[9]=y[9];

  for (;;) {
    hh=0.5*h;
    rk4(ysav,dysav,xsav,hh,ytemp,derivs);
    *x=xsav+Cinc(hh);
    (*derivs)(dydx,(int) *x,ytemp);
    rk4(ytemp,dydx,*x,hh,y,derivs);
    *x=xsav+Cinc(h);
    if (*x == xsav)
      return(0); /* Convergence Failure. */
    rk4(ysav,dysav,xsav,h,ytemp,derivs);
    errmax=0.0;

    ytemp[0] -= y[0];
    if (errmax < (temp=fabs(ytemp[0]/yscal[0]))) errmax=temp;
    ytemp[1] -= y[1];
    if (errmax < (temp=fabs(ytemp[1]/yscal[1]))) errmax=temp;
    ytemp[2] -= y[2];
    if (errmax < (temp=fabs(ytemp[2]/yscal[2]))) errmax=temp;
    ytemp[3] -= y[3];
    if (errmax < (temp=fabs(ytemp[3]/yscal[3]))) errmax=temp;
    ytemp[4] -= y[4];
    if (errmax < (temp=fabs(ytemp[4]/yscal[4]))) errmax=temp;
    ytemp[5] -= y[5];
    if (errmax < (temp=fabs(ytemp[5]/yscal[5]))) errmax=temp;

    if ((errmax /= eps) <= 1.0) {
      *hdid=h;
      *hnext=(errmax > ERRCON ? SAFETY*h*pow(errmax, PGROW) : 4.0*h);
      break;
    }
    h=SAFETY*h*exp(PSHRINK*log(errmax));
  }

  y[0] += ytemp[0]*FCOR;
  y[1] += ytemp[1]*FCOR;
  y[2] += ytemp[2]*FCOR;
  y[3] += ytemp[3]*FCOR;
  y[4] += ytemp[4]*FCOR;
  y[5] += ytemp[5]*FCOR;
  return(1); /* success */
}

#undef PGROW
#undef PSHRINK
#undef FCOR
#undef SAFETY
#undef ERRCON
```

simulate.c

```
/* Runge--Kutta simulation of cardiovascular hemodynamics.
 * Timothy Davis, May 1989.  Previous versions by R.Sah and G.Moody.
```

```
 * Copyright 1988, 1989 Timothy L. Davis
 * $Source: /usr/lut/src/sim/RCS/simulate.c,v $
 *
 * Run the CV simulation for a specific number of cycles.  Writes simulation
 * data to extern procedure SimDataEnq: (6 pressures + time + Cl + Cr +
 * capacitance array index)
 * simulate() takes original pressure/time/capacitance vector (x),& # of beats.
 * This simulator expects parameters to be set up (equs.c:initequvars()),
 * and for initialization of the pressure array via estimate.c:estimate() or
 * previous call to simulate().
 */
#include <signal.h>
#include <math.h>
#include "CVDefs.h"
#include "../event/Event.h"
#define ADAPT_RK /* Use adaptive step size (quality control) Runge-Kutta */

extern double Tsystole(), Cl(), Cr(), Cinc(); /* all from file equs.c */
extern double getval();
extern void equs();

/*PUBLIC*/
int simulate(x, ticks)
double x[]; /* starting values of 6 pressures, time, Cl, Cr, and
                        current index into lvlv capacitance table */
int ticks; /* number of consecutive times steps desired */
{
  int j;
  double xsav[10]; /* private copy of pressure data for speed */
  double dxdt[6];                       /* dx/dt[], evaluated by "equs" */
#ifndef ADAPT_RK
  double dt;                            /* time increment */
#endif ADAPT_RK
#ifdef SUN
  void (*old_handler)()=signal(SIGFPE, fpe_in_simulate); /* from Event.c */
#else assuming BSD
  int (*old_handler)()=signal(SIGFPE, fpe_in_simulate); /* from Event.c */
#endif SUN

  for (j=0; j<10; j++) xsav[j] = x[j];

  for (j = 0; j < ticks; j++) { /* do ticks Runge-Kutta segments */
    initvalves(x);          /* set valves depending on pressures */
    fixvolume(x);                       /* correct at eye venous reservoir */

#ifndef ADAPT_RK
    dt =  6*Tsystole(getval(x, SYSTEM, HEARTRATE))/SYSTOLESEGS;
    samode(x, dt); /* determine onset of contraction and update constants */
    equs(dxdt, (int) x[9], x);
    rk4(x, dxdt, x[9], dt, x, equs);
    x[6] += dt;
    x[9] += Cinc(dt);
    x[7] = Cl((int) x[9]);
    x[8] = Cr((int) x[9]);
    if (false(x[9])<1000 && false(x[1])<1000 && false(x[2])<1000
          && false(x[3])<1000 && false(x[4]) < 1000 && false(x[5]) < 1000)
      SimDataEnq(x);
    else {
      signal(SIGFPE,old_handler);
      for (j=0; j<10; j++) x[j] = xsav[j];
      return(0); /* FALSE to restart simulation attempt */
    }
#else ADAPT_RK
    {
      static double hdid = 0.000;
      static double htry = 0.003;
      static double yscale[] = {1, 1, 1, 1, 1, 1, 1};
      double hnext;

      samode(x, hdid); /* determine onset of contraction */
      equs(dxdt, (int) x[9], x);
      if (!rkqc(x, dxdt, &x[9], htry, 0.01,
                yscale, &hdid, &hnext, equs)) {
        signal(SIGFPE,old_handler);
        for (j=0; j<10; j++) x[j] = xsav[j];
        return(0); /* Convergence failure! */
      }
      htry = hnext > 0.01  ? 0.01 : hnext; /* Keep arterial diastole smooth */
      x[6] += hdid;
      x[7] = Cl((int)x[9]);
      x[8] = Cr((int)x[9]);
      SimDataEnq(x);
    }
#endif ADAPT_RK
  }
  signal(SIGFPE,old_handler);
  return(1); /* successful... done */
}
```

C.3 Database parsing and simulation initialization

These files handle the reading of the database file `cvLib/parameter.db` , which contains default names and limits on all independent parameters and dependent variables in human-readable form. It also handles the initialization of the patient case environment from one of the patient description files `Patients/hist.?` .

Parameter.h

```
/* $Header: /usr/lut/src/data/RCS/Parameter.h,v 1.1 90/08/18 18:30:38 tldavis Exp $
 * Parameter.h:  Timothy L. Davis, 1990
 *
 * Header for simulation parameter database and patient database routines.
 * These are defined in parameter.c, patient.c, and format.c
 * $Log:            Parameter.h,v $
 * Revision 1.1  90/08/18  18:30:38  tldavis
 * Initial revision
 *
 */

/*
 * parameter.c public functions:
 */

extern void LoadParameterData(/* char *filename;*/);
extern SIMULATION *getsimulation(/* int pathology_number; */);
extern SIMULATION *setsimulation();
extern int var_in_bounds(/*int place, type, value;*/);
extern int *get_varlist(/*int place, device, dep;*/);
```

```
extern VarData *get_vardata(/*int place, type;*/);

/*
 * patient.c:  Predefined clinical patients with case history.
 * LoadPatient loads the patient simulation and returns the case history.
 */
extern char *LoadPatient(/*char *path;*/);

/*
 * format.c:  Formats the parameter list for printing.  Returns the list.
 * Parameters different from default are marked as such.
 */
extern char *FormatParams(/* SIMULATION *presentsim, *defaultsim */);
```

format.c

```
/*
 *      $Source: /mit/hst/src/data/RCS/format.c,v $
 */
#ifndef lint
static char *rcs="$Header: /mit/hst/src/data/RCS/format.c,v 2.1 90/08/18 18:31:02 tldavis Exp $";
#endif  lint

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "../sim/CVDefs.h"
#include "Parameter.h"

extern char *malloc();

static char *page[] = {
  "Heart Rate:    %4.2lf%3s\n",
  "Total Blood Volume:    %6.1lf%3s\n",
  "Trans-thoracic Pressure:    %4.2lf%3s\n",

  "\nCapacitances:    (ml/mmHg)\n----------\n",
  "LVsyst %4.2lf%3s LVdiast %4.2lf%3s Arterial %4.2lf%3s Venous %4.2lf%3s\n",
  "RVsyst %4.2lf%3s RVdiast %4.2lf%3s Pulm Art %4.2lf%3s Pulm V %4.2lf%3s\n",

  "\nZero-Pressure Volumes:    (ml)\n----------\n",
  "LV %6.1lf%3s Arterial %6.1lf%3s Venous %6.1lf%3s\n",
  "RV %6.1lf%3s Pulm Art %6.1lf%3s Pulm V %6.1lf%3s\n",

  "\nResistances:    (mmHg*sec/ml)\n----------\n",
  "LV Inflow %4.2lf%3s LV Outflow %4.2lf%3s Microvascular %4.2lf%3s\n",
  "Venous     %4.2lf%3s RV Outflow %4.2lf%3s Pulmonary      %4.2lf%3s\n",
  };

static char *pm(a, b)
double a, b;
{ return (a > b ? "(+)" : (a < b ? "(-)" :  "   "));
}

/*PUBLIC*/ char *FormatParams(n, o)
SIMULATION *n, *o;
{
  static char line[512], buf[10000];

  buf[0] = '\0';
  sprintf(line,page[0], n->hr, pm(n->hr,o->hr)); strcat(buf, line);
  sprintf(line,page[1], n->bv, pm(n->bv,o->bv)); strcat(buf, line);
  sprintf(line,page[2], n->pth, pm(n->pth,o->pth)); strcat(buf, line);
  sprintf(line,page[3]); strcat(buf, line);
  sprintf(line,page[4],
          n->Clsys,pm(n->Clsys,o->Clsys), n->Cldias,pm(n->Cldias,o->Cldias),
          n->Ca,pm(n->Ca,o->Ca),n->Cv,pm(n->Cv,o->Cv));
  strcat(buf, line);
  sprintf(line,page[5],
          n->Crsys,pm(n->Crsys,o->Crsys), n->Crdias,pm(n->Crdias,o->Crdias),
          n->CPa,pm(n->CPa,o->CPa),n->CPv,pm(n->CPv,o->CPv));
  strcat(buf, line);
  sprintf(line,page[6]);
  strcat(buf, line);
  sprintf(line,page[7], n->Vlt0, pm(n->Vlt0,o->Vlt0),
          n->Va0,  pm(n->Va0,o->Va0),   n->Vv0,  pm(n->Vv0,o->Vv0));
  strcat(buf, line);
  sprintf(line,page[8], n->Vr0, pm(n->Vr0,o->Vr0),
          n->VPa0, pm(n->VPa0,o->VPa0), n->VPv0, pm(n->VPv0,o->VPv0));
  strcat(buf, line);
  sprintf(line,page[9]);
  strcat(buf, line);
  sprintf(line,page[10], n->Rli,pm(n->Rli,o->Rli), n->Rlo,pm(n->Rlo,o->Rlo),
          n->Ra,pm(n->Ra,o->Ra));
  strcat(buf, line);
  sprintf(line,page[11], n->Rv,pm(n->Rv,o->Rv),   n->Rro,pm(n->Rro,o->Rro),
          n->RPv,pm(n->RPv,o->RPv));
  strcat(buf, line);

#ifdef DEBUG
  printf("SIMULATION PARAMETERS:\n\n %s\n", buf); fflush(stdout);
#endif
  return(strcpy(malloc(strlen(buf)+1),buf));
}
```

patient.c

```
/*
 *$Source: /mit/hst/src/data/RCS/patient.c,v $
 */

#ifndef lint
static char *rcs="$Header: /mit/hst/src/data/RCS/patient.c,v 2.2 90/08/18 18:31:30 tldavis Exp $";
#endif  lint

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "../sim/CVDefs.h"
#include "Parameter.h"

/*************************************
 *
 *  Public Routines
 *
 * ONLY ONE VARDATA STRUCTURE.  RESET BEFORE EACH PATIENT CHANGE
 *************************************/

/*PUBLIC*/
char *LoadPatient(path)
char *path;
```

```
{
 FILE *fp;
 static char line[256], *p, history[10000];
 int placeCode = -1, typeCode = -1;
 double value;
 VarData *vdPtr;

 history[0] = '\0';
 if ((fp = fopen(path, "r")) == NULL) {
  fprintf(stderr, "Can't open patient file %s.\n",path);
  exit(1);
 }

 /* discard leading comments*/
 while (fgets(line, sizeof(line), fp) != NULL)
  if (line[0] != '#') break;

 /* read history up until comment */
 do
  strcat(history, line);
 while (fgets(line, sizeof(line), fp) != NULL && line[0] != '#');

 /* read parameter updates */
 while (fgets(line, sizeof(line), fp) != NULL) {
  if (line[0] == '#') continue; /* discard comments from now on */
  if (line[0] == '\0' || line[1] == '\0') break;
  p = line; while (!isalpha(*p)) p++;

  if (strncmp(p, "l", 1) == 0) placeCode = LV;
  else if (strncmp(p, "r", 1) == 0) placeCode = RV;
  else if (strncmp(p, "a", 1) == 0) placeCode = SA;
  else if (strncmp(p, "v", 1) == 0) placeCode = SV;
  else if (strncmp(p, "pa",2) == 0) placeCode = PA;
  else if (strncmp(p, "pv",2) == 0) placeCode = PV;
  else if (strncmp(p, "micro",1)==0) placeCode = SC;
  else if (strncmp(p, "pmicro",2)==0)placeCode = PC;
  else if (strncmp(p,"sys",3) == 0) placeCode = SYSTEM;
  else {fprintf(stderr,"Bad Place %s in file %s.\n", p, path);
        exit(1);}
  while (isalpha(*p)) p++;
  while (isspace(*p)) p++;

  if (strncmp(p, "csys", 4) == 0) typeCode = CAPACITANCESYS;
  else if (strncmp(p, "cdias", 4) == 0) typeCode = CAPACITANCEDIAS;
  else if (strncmp(p, "cap", 3) == 0) typeCode = CAPACITANCE;
  else if (strncmp(p, "zvol",4) == 0) typeCode = ZPVOLUME;
  else if (strncmp(p, "zp", 2) == 0)  typeCode = ZPVOLUME;
  else if (strncmp(p, "res", 3) == 0) typeCode = RESISTANCE;
  else if (strncmp(p, "hr", 2) == 0) typeCode = HEARTRATE;
  else if (strncmp(p, "pres", 4) == 0) typeCode = PRESSURE;
  else if (strncmp(p, "vol", 3) == 0) typeCode = VOLUME;
  else if (strncmp(p, "flow", 4) == 0) typeCode = FLOW;
  else if (strncmp(p, "time", 4) == 0) typeCode = TIME;
  else {fprintf(stderr,"Bad Type %s in file %s.\n", p, path);
        exit(1);}
  while (isalpha(*p)) p++;
  while (isspace(*p)) p++;

  sscanf(p,"%lf", &value);

  vdPtr = get_varlists(placeCode, typeCode);

  #ifdef DEBUG
    printf("Patient:   place %d type %d old value %lf new value %lf\n",placeCode,
          typeCode, vdPtr->normal, value);
  #endif
    vdPtr->normal = value;
  }

  return(history);
}
```

test.parameter.c

```
/*
 *      $Source: /mit/hst/src/data/RCS/test.parameter.c,v $
 *      $Header: /mit/hst/src/data/RCS/test.parameter.c,v 2.0 90/08/06 17:51:04 tldavis Stab $
 */

#ifndef lint
static char *rcsid_test_parameter_c = "$Header: /mit/hst/src/data/RCS/test.parameter.c,v 2.0 90/08/06 17:51:04 tldavis Stab $";
#endif  lint

#include "../include/cvdefs.h"

main()
{
 extern int *get_varlist();
 int *v, place, type, n;

 LoadParameterData("db");
 for (;;) {
  printf("place, type?\n");
  scanf("%d %d", &place, &type);
  v = get_varlist (place, type, DEPENDENT);
  printf("ENTERED: %d %d \n", place, type);
  for (n=0; v[n] != ENDVAR; n++)
   printf("DEPENDENT var %d = %d \n", n, v[n]);
  v = get_varlist (place, type, INDEPENDENT);
  for (n=0; v[n] != ENDVAR; n++)
   printf("INDEPENDENT var %d = %d \n", n, v[n]);
 }
}
```

C.4 NumberLine widget

These files define the NumberLine widget, which displays a value on a continuous number line, and allows the user to change it by moving a cross-hair over the line and clicking or typing and hitting the return key. Events are produced immediately for every parameter-change event. See `NumberLine.h` for a list of resources available for interfacing with user programs.

InputP.h

```
#ifndef _InputP_h
#define _InputP_h
/*****************************************************************************
 * $Header: /mit/hst/src/nl/RCS/InputP.h,v 2.1 90/08/10 10:58:49 dot Exp $
 * $Source: /mit/hst/src/nl/RCS/InputP.h,v $
 *
 * Input Widget Private Data Definitions
 *
```

```
**********/

#include <X11/Xaw/SimpleP.h>

typedef struct {
  int foo;
} InputClassPart;

typedef struct _InputClassRec {
  CoreClassPart      core_class;
  SimpleClassPart    simple_class;
  InputClassPart     input_class;
} InputClassRec;

extern InputClassRec inputClassRec;
extern WidgetClass   inputWidgetClass;

typedef struct {
  int foo;
} InputPart;


/**********************************************************************
 *
 * Full instance record declaration
 *
 **********************************************************************/

typedef struct _InputRec {
    CorePart       core;
    SimplePart     simple;
    InputPart      input;
} InputRec, *InputWidget;

#endif _InputP_h
```

NumberLine.h

```
#ifndef _NumberLine_h
#define _NumberLine_h
/**********************************************************************
 *
 * Number Line Widget
 * X11 Version
 * Copyright 1988 Timothy L. Davis
 *
 * $Source: /usr/lsa/src/xl/RCS/NumberLine.h,v $
 * $Header: /usr/lsa/src/xl/RCS/NumberLine.h,v 2.2 90/08/18 18:35:51 tldavis Exp $
 **********************************************************************/

#ifdef XAW_BC
#define XtNlabel                "label"
#define XtNfont                 "font"
#endif XAW_BC
#define XtNminimum              "minimum"
#define XtNmaximum              "maximum"
#define XtNpresent              "present"
#define XtNdefault              "default"
#define XtNprevious             "previous"
#define XtNdone                 "done"
#define XtNpercentage           "percentage"

#ifndef XtNcallback
#define XtNcallback             "callback"
#endif  XtNcallback

extern WidgetClass numberLineWidgetClass;

extern void NumberLineGetValue(); /* w, value, done */
 /* Widget w;
  * double *value;
  * int *done;
  */

extern Boolean NumberLineEnroll(); /* w, place, type */
 /* Widget w;
  * int place;
  * int type;
  */

extern void NumberLineCancelType(); /* w */
 /* Widget w;
  */

extern void NumberLineResetAll();
extern void NumberLineDestroyAll();
extern Boolean NumberLineCheckType(); /*int place, int type */
/*
 * Format for Callback procedure:
 *
 *   void Proc(w, client_data, call_data)
 *       Widget w;              * Widget for which callback is registered
 *       caddr_t client_data;   * client-specified data closure
 *       caddr_t call_data;     * unused
 *
 * Add callbacks using XtAddCallback of type XtNcallback
 */

#endif _NumberLine_h
```

NumberLineP.h

```
#ifndef _NumberLineP_h
#define _NumberLineP_h
/**********************************************************************
 *
 * NumberLine Widget Private Data Definitions
 *
 **********************************************************************/

#include <X11/CompositeP.h>
#include "NumberLine.h"

#define XtRDouble               "double"

typedef struct {
  GC gc;
  XFontStruct *big_font, *small_font;
  Cursor pointer_cursor;
```

```
  Cursor invisible_cursor;
} NumberLineClassPart;

typedef struct _NumberLineClassRec {
  CoreClassPart        core_class;
  CompositeClassPart   composite_class;
  NumberLineClassPart  numberLine_class;
} NumberLineClassRec;

extern NumberLineClassRec numberLineClassRec;

typedef struct {
  /*resources*/
  char *title;
  double min, max;
  double pres, def;
  double prev; /* value at previous call */
  int done;
  int percentage;
  Pixel foreground, background;

  /* private state */
  XtCallbackList callbacks;
  double head_val, ptr_val; /* to keep track of pointer and pointer head */
  double sf, gran; /* scaling factor and granularity of pointer */
  int old_ptr_x, old_ptr_y, old_hd_x, old_hd_y;
  int minpt, maxpt, yline, *iccins; /* minimum and maximum coordinates, ypos.*/
  int ymmax; /* y axis for numbers */
  Widget target_w; /*this is the window which receives mousemoved events*/
  Widget def_w, prev_w, done_w, key_w, title_w; /* all widgets */
  char *string;
} NumberLinePart;


/**********************************************************************
 *
 * Full instance record declaration
 *
 **********************************************************************/

typedef struct _NumberLineRec {
    CorePart        core;
    CompositePart compositie;
    NumberLinePart          numberLine;
} NumberLineRec, *NumberLineWidget;

#endif _NumberLineP_h
```

Input.c

```
#ifndef lint
static char *sccs="$Header: /usr/lust/src/nl/RCS/Input.c,v 2.1 90/08/10 16:14:19 tldavis Exp $";
#endif  lint
/**********************************************************************
 * Input widget is a shell of a widget, does nothing but
 * create a window of type InputOnly.
 * $Source: /usr/lust/src/nl/RCS/Input.c,v $
 * $Log:            Input.c,v $
 * Revision 2.1  90/08/10  16:14:19  tldavis
 * Added a line to the SimplePart declaration for X11R4.
 *
 * Revision 2.0  90/08/06  17:57:29  tldavis
 * Final X11R3 version.
 *
 * Revision 1.4  89/02/22  12:26:20  tldavis
 * X11R3 version.
 *
 * Revision 1.3  88/08/23  11:19:15  tldavis
 * Cleaned up header file references.
 *
 **********************************************************************/
#include <stdio.h>
#include <X11/IntrinsicP.h>
#include <X11/StringDefs.h>
#include "InputP.h"

/*ARGSUSED*/ static void InputOnlyRealize(w,mask,attributes)
    Widget w;
    Mask *mask; /* unused */
    XSetWindowAttributes *attributes;
{
  Mask inputMask = CWWinGravity | CWEventMask | CWDontPropagate |
                   CWOverrideRedirect | CWCursor;
  w->core.border_width = 0;
  w->core.depth = 0; /* InputOnly window MUST have these == 0 */
  XtCreateWindow(w, (unsigned int)InputOnly, (Visual *) CopyFromParent,
                 (Mask)(*mask & inputMask) , attributes);
}

InputClassRec inputClassRec = {
  { /* superclass            */    (WidgetClass) &simpleClassRec,
    /* class_name            */    "Input",
    /* widget_size           */    sizeof(InputClassRec),
    /* class_initialize      */    NULL,
    /* class_part_initialize */    NULL,
    /* class_inited          */    FALSE,
    /* initialize            */    NULL,
    /* initialize_hook       */    NULL,
    /* realize               */    InputOnlyRealize,
    /* actions               */    NULL,
    /* num_actions           */    0,
    /* resources             */    NULL,
    /* num_resources         */    0,
    /* xrm_class             */    NULLQUARK,
    /* compress_motion       */    TRUE,
    /* compress_exposure     */    TRUE,
    /* compress_enterleave   */    TRUE,
    /* visible_interest      */    FALSE,
    /* destroy               */    NULL,
    /* resize                */    XtInheritResize,
    /* expose                */    NULL,
    /* set_values            */    NULL,
    /* set_values_hook       */    NULL,
    /* set_values_almost     */    XtInheritSetValuesAlmost,
    /* get_values_hook       */    NULL,
    /* accept_focus          */    NULL,
    /* version               */    XtVersion,
    /* callback_private      */    NULL,
    /* tm_table              */    NULL,
    /* query_geometry        */    XtInheritQueryGeometry,
    /* NEW R3 display_accelerator*/ XtInheritDisplayAccelerator,
```

```
    /* NEW R3 extension */          NULL,
  },
  { /* Simple fields */
#ifndef XAW_BC
    /* NEW R4 change sensitive */   XtInheritChangeSensitive,
#else /* no XAW_BC */
                                    0,
#endif XAW_BC
  },
  { /* Input fields */ 0, },
};
WidgetClass inputWidgetClass = (WidgetClass) &inputClassRec;
```

NumberLine.c

```
#ifndef lint
static char
*rcsv="$Header: /mit/hut/src/nl/RCS/NumberLine.c,v 2.4 90/08/28 14:40:11 tldavis Exp $";
#endif  lint
/*****************************************************
 *
 * Number Line widget for X11R4 Cardiovascular Simulator
 * Copyright 1989 by Timothy L. Davis
 * For (probable) X11R3 compatibility, define symbol XAWBC.
 *
 * $Source: /mit/hut/src/nl/RCS/NumberLine.c,v $
 * $Log:    NumberLine.c,v $
 * Revision 2.4  90/08/28  14:40:11  tldavis
 * Fixed a bug with the text widget, which needs a new string for each change.
 *
 * Revision 2.3  90/08/18  18:35:11  tldavis
 * Removed coloring to do it with a resource file instead.
 * General cleanup.
 *
 * Revision 2.2  90/08/10  16:14:50  tldavis
 * Fixed asciiText text replacement for X11R4.
 *
 * Revision 2.1  90/08/10  10:55:49  dst
 * upgrade to R4; changes to the asciiText widgets
 * now code ifdef'd under !XAW_BC
 *
 * Revision 2.0  90/08/06  17:57:37  tldavis
 * Final X11R3 version.
 *
 * Revision 1.13  90/08/06  17:06:27  tldavis
 * Fixed problems with color on Sons.
 *
 * Revision 1.12  90/05/31  16:57:15  tldavis
 * Pre-ACES checkin.
 *
 * Revision 1.11  89/10/12  22:15:00  tldavis
 * Updated fall 1989.
 *
 * Revision 1.10  89/09/02  01:03:10  tldavis
 * Made width dep. on title length work.
 *
 * Revision 1.9  89/09/02  00:37:40  tldavis
 * Made pointer even prettier, made width dep. on label.
 *
 * Revision 1.8  89/09/01  23:39:51  tldavis
 * Made pointer prettier, widget smaller, more compact, added -- key activity.
 *
 * Revision 1.6  89/06/13  21:24:06  tldavis
 * R3 fonts fixed.
 * Revision 1.5  89/02/22  12:26:30  tldavis
 * X11R3 version.
 * Revision 1.1  88/10/10  22:30:50  hut
 * Initial revision -- previous revisions deleted.
 *****************************************************/

#include <stdio.h>
#include <math.h>
#include <X11/IntrinsicP.h>
#include <X11/ShellP.h>
#include <X11/StringDefs.h>
#include <X11/Xaw/LabelP.h> /* necessary for label to redisplay on demand. */
#include <X11/Xaw/Command.h>
#include <X11/Xaw/AsciiText.h>
#ifdef XAW_BC
#include <X11/Xaw/TextP.h> /* necessary to get XfTextBlock definition! */
#endif XAWBC
#include <X11/cursorfont.h>
#include "NumberLineP.h"
#include "InputP.h"

#define NL_MAX_TICS 10
#define NL_HDR_HEIGHT 30

extern Display *dpy;
extern Window  root;
extern int screen;

/*****************************************************
 *
 * Full class record constant
 *
 *****************************************************/

/* There is a problem with lc on the RT when compiling the XtOffset macro.
 * Thought this would work better, but... "illegal initialization" to pcc ...
 */
#define offset(field) (XtOffset(NumberLineWidget,numberLine.field))

static XtResource resources[] = {
  {XtNforeground, XtCForeground, XtRPixel, sizeof(Pixel),
    offset(foreground), XtRString, "XtDefaultForeground"},
  {XtNbackground, XtCBackground, XtRPixel, sizeof(Pixel),
    offset(background), XtRString, "XtDefaultBackground"},
  {XtNlabel, XtCLabel, XtRString, sizeof(String),
    offset(title), XtRString, (caddr_t)"Number Line Value Choice"},
  {XtNminimum, XtCValue, XtRDouble, sizeof(double),
    offset(min), XtRString, (caddr_t)"0.0"},
  {XtNmaximum, XtCValue, XtRDouble, sizeof(double),
    offset(max), XtRString, (caddr_t)"100.0"},
  {XtNpresent, XtCValue, XtRDouble, sizeof(double),
    offset(pres), XtRString, (caddr_t)"50.0"},
  {XtNdefault, XtCValue, XtRDouble, sizeof(double),
```

```
    offset(def), XtRString, (caddr_t)"30.0"},
  {XtNprevious, XtCValue, XtRDouble, sizeof(double),
    offset(prev), XtRString, (caddr_t)"50.0"},
  {XtNdone, XtCValue, XtRInt, sizeof(int),
    offset(done), XtRString, (String)NULL},
  {XtNpercentage, XtCValue, XtRInt, sizeof(int),
    offset(percentage), XtRString, (caddr_t)"0"},
  {XtNcallback, XtCCallback, XtRCallback, sizeof(caddr_t),
    offset(callbacks), XtRCallback, (caddr_t)NULL},
};
#undef offset

/*
 * FORWARD defines for the widget activity routines to be put in core struct
 * These routines follow widget interaction rules.
 */
static void ClassInitialize();
static void Initialize();
static void Realize();
static void Destroy();
static void Redisplay();
static void GetValuesHook();
static XtGeometryResult GeometryHandler();

NumberLineClassRec numberLineClassRec = {
  { /* core_class fields */
#define superclass ((WidgetClass) &compositeClassRec)
    /* superclass         */    superclass,
    /* class_name         */    "NumberLine",
    /* widget_size        */    sizeof(NumberLineRec),
    /* class_initialize   */    ClassInitialize,
    /* class_part_initialize */ NULL,
    /* class_inited       */    False,
    /* initialize         */    Initialize,
    /* initialize_hook    */    NULL,
    /* realize            */    Realize,
    /* actions            */    NULL,
    /* num_actions        */    0,
    /* resources          */    resources,
    /* num_resources      */    XtNumber(resources),
    /* xrm_class          */    NULLQUARK,
    /* compress_motion    */    True,
    /* compress_exposure  */    True,
    /* compress_enterleave */   True,
    /* visible_interest   */    False,
    /* destroy            */    Destroy,
    /* resize             */    XtInheritResize,
    /* expose             */    Redisplay,
    /* set_values         */    NULL,
    /* set_values_hook    */    NULL,
    /* set_values_almost  */    XtInheritSetValuesAlmost,
    /* get_values_hook    */    GetValuesHook,
    /* accept_focus       */    NULL,
    /* version            */    XtVersion,
    /* callback_private   */    NULL,
    /* tm_table           */    NULL,
    /* query_geometry     */    XtInheritQueryGeometry,
        /* NEW#3 */              XtInheritDisplayAccelerator,
        /* NEW#3 */              NULL,
  },
  { /* composite_class fields */
    /* geometry_manager */   GeometryHandler,
    /* change_managed */     XtInheritChangeManaged,
    /* insert_child */       XtInheritInsertChild,
    /* delete_child */       XtInheritDeleteChild,
    /* ***NEW#3** extension*/ NULL,

  },
  { /* NumberLine fields */
    /* gc */                 (GC) NULL,
    /* big_font */           (XFontStruct *) NULL,
    /* small_font */         (XFontStruct *) NULL,
    /* pointer_cursor */     (Cursor) NULL,
    /* invisible_cursor */   (Cursor) NULL,
  }
};
WidgetClass numberLineWidgetClass = (WidgetClass) &numberLineClassRec;

/*****************************************************************
 *
 * Private Procedures
 *
 *****************************************************************/
/*FORWARD*/static void DefaultCallback(),PreviousCallback(),
                TargetHandler(), KeyHandler();

/*ARGSUSED*/static void CvtStringToDouble(args, num_args, fromVal, toVal)
XrmValue *args;
Cardinal *num_args;
XrmValue *fromVal, *toVal;
{
  static double i;
  /* is this dangerous??  can we guarantee that the toolkit will copy
   * the return value before calling this routine again? (I hope so!)
   */
  if (*num_args != 0)
    XtWarning("String to Double conversion needs no extra arguments");
  if (sscanf ((char *)fromVal->addr,"%lf",&i) == 1) {
    toVal->size = sizeof(double);
    toVal->addr = (caddr_t) &i;
  }
}

/*
 * registered private widget initialization and event handling routines:
 *
 */
static void ClassInitialize ()
{
  XFontStruct *font;
  static XtColor color = { 1, 0, 0, 0, 0, 0};
  XGCValues gcv;
  unsigned long fontmask = 0;

  if (font = XLoadQueryFont(dpy,"*-new century*-bold-r-*-*-*-120-*"))
    numberLineClassRec.numberLine_class.big_font = font;

  if (numberLineClassRec.numberLine_class.small_font =
      XLoadQueryFont(dpy,"*-helvetica-medium-r-*-*-*-80-*"))
    fontmask = GCFont;
```

```
 if (fontmask)
   gcv.font = numberLineClassRec.numberLine_class.small_font->fid;
 numberLineClassRec.numberLine_class.gc =
   XCreateGC(dpy, root, fontmask /*| GCForeground | GCBackground */, &gcv);

 if (font)
   numberLineClassRec.numberLine_class.invisible_cursor =
     XCreateGlyphCursor(dpy, font->fid, font->fid, (unsigned int)' ',
                        (unsigned int)' ', &color, &color);
 else
   numberLineClassRec.numberLine_class.invisible_cursor =
     XCreateFontCursor(dpy, XC_crosshair);
 numberLineClassRec.numberLine_class.pointer_cursor =
   XCreateFontCursor(dpy, XC_arrow);
 XtAddConverter(XtRString, XtRDouble, CvtStringToDouble, (ArgList)NULL, 0);
} /* ClassInitialize() */

/*ARGSUSED*/
static void Initialize(request, new)
    Widget request, new;
{
 NumberLineWidget nlw = (NumberLineWidget) new;
 NumberLinePart *nl = &( nlw->numberLine);
 int i;
 Arg blist[20];
 Pixel fore = nl->foreground;
 Pixel back = nl->background;

 /* The first 10 elements represent number key presses; the intent of this
  * translation table is to allow only numeric entry on only 1 line of text.
  */
 static String nullEventTable =
  "<Key>0x030:       kill-to-end-of-line() beginning-of-file() end-of-line() \
             insert-char() \n\
     <Key>0x031:       kill-to-end-of-line() beginning-of-file() end-of-line() \
             insert-char() \n\
     <Key>0x032:       kill-to-end-of-line() beginning-of-file() end-of-line() \
             insert-char() \n\
     <Key>0x033:       kill-to-end-of-line() beginning-of-file() end-of-line() \
             insert-char() \n\
     <Key>0x034:       kill-to-end-of-line() beginning-of-file() end-of-line() \
             insert-char() \n\
     <Key>0x035:       kill-to-end-of-line() beginning-of-file() end-of-line() \
             insert-char() \n\
     <Key>0x036:       kill-to-end-of-line() beginning-of-file() end-of-line() \
             insert-char() \n\
     <Key>0x037:       kill-to-end-of-line() beginning-of-file() end-of-line() \
             insert-char() \n\
     <Key>0x038:       kill-to-end-of-line() beginning-of-file() end-of-line() \
             insert-char() \n\
     <Key>0x039:       kill-to-end-of-line() beginning-of-file() end-of-line() \
             insert-char() \n\
     <Key>.:    kill-to-end-of-line() beginning-of-file() end-of-line() \
             insert-char() \n\
     <Key>-:    kill-to-end-of-line() beginning-of-file() end-of-line() \
             insert-char() \n\
     Ctrl<Key>F:     forward-character() \n\
     <Key>0xff53:    forward-character() \n\
     Ctrl<Key>B:     backward-character() \n\
     <Key>0xff51:    backward-character() \n\
     Ctrl<Key>A:     beginning-of-line() \n\
     Ctrl<Key>E:     end-of-line() \n\
     Ctrl<Key>P:     previous-line() \n\
     <Key>0xff52:    previous-line() \n\
     Meta<Key>\\<:   beginning-of-file() \n\
     Ctrl<Key>Z:     scroll-one-line-up() \n\
     Ctrl<Key>D:     delete-next-character() \n\
     Ctrl<Key>H:     delete-previous-character() \n\
     <Key>0xff7f:    delete-previous-character() \n\
     <Key>0xffff:    delete-previous-character() \n\
     <Key>0xff08:    delete-previous-character() \n\
     Meta<Key>D:     delete-next-word() \n\
     Meta<Key>H:     delete-previous-word() \n\
     Shift Meta<Key>D: kill-word() \n\
     Shift Meta<Key>H: backward-kill-word() \n\
     Ctrl<Key>W:     kill-selection() \n\
     Ctrl<Key>K:     kill-to-end-of-line() \n\
     Meta<Key>K:     kill-to-end-of-paragraph()";

#ifdef DEBUG
 XSync(dpy, False);
 fprintf(stderr, "NumberLine.Initialize...\n");
#endif DEBUG
 nl->ticsize = 5;

 if (nl->percentage) {
   nl->min = 10;
   nl->max = 200;
   nl->press *= 100./nl->def;
   if (nl->prev) nl->prev *= 100./nl->prev;
 }

 XtSetState(dpy, numberLineClassRec.numberLine_class.gc, fore, back, GXcopy,
            AllPlanes);
 i = 0;
 XtSetArg(blist[i],XtNborderColor, fore);i++;
 if (numberLineClassRec.numberLine_class.big_font)
   XtSetArg(blist[i],XtNfont,
            numberLineClassRec.numberLine_class.big_font);i++;
 nl->def_w = XtCreateManagedWidget("Default",commandWidgetClass, nlw,blist,i);
 XtAddCallback(nl->def_w,XtNcallback,DefaultCallback, nlw);

 i = 0;
 XtSetArg(blist[i],XtNborderColor, fore);i++;
 if (numberLineClassRec.numberLine_class.big_font)
   XtSetArg(blist[i],XtNfont,
            numberLineClassRec.numberLine_class.big_font);i++;
 nl->prev_w=XtCreateManagedWidget("Previous",commandWidgetClass,nlw,blist, i);
 XtAddCallback(nl->prev_w,XtNcallback, PreviousCallback, nlw);

 i = 0;
 XtSetArg(blist[i], XtNlabel, nl->title);i++;
 XtSetArg(blist[i],XtNborderWidth, 0);i++;
 if (numberLineClassRec.numberLine_class.big_font)
   XtSetArg(blist[i],XtNfont,
            numberLineClassRec.numberLine_class.big_font);i++;
 nl->title_w = XtCreateManagedWidget("label",labelWidgetClass, nlw, blist, i);

 i = 0;
 XtSetArg(blist[i],XtNwidth, 10);i++;
 XtSetArg(blist[i],XtNheight, 10);i++;
```

```
  nl->target_w = XtCreateManagedWidget("target",inputWidgetClass,nlw,blist, i);

  nl->string = (char *)XtMalloc(32*sizeof(char));
  i=0;
#ifdef XAW_BC
  XtSetArg(blist[i],XtNeditType, XttextEdit);i++;
#else /* ! XAW_BC */
  XtSetArg(blist[i],XtNeditType, XawtextEdit);i++;
#endif /* XAW_BC */
  XtSetArg(blist[i],XtNstring, nl->string);i++;
  XtSetArg(blist[i],XtNlength, 10);i++;
#ifdef XAW_BC
  XtSetArg(blist[i],XtNtextOptions, editable); i++;
#endif /* XAW_BC */
  XtSetArg(blist[i],XtNwidth, 80);i++;
  XtSetArg(blist[i],XtNtranslations,XtParseTranslationTable(textEventTable));
  i++;
  if (numberLineClassRec.numberLine_class.big_font)
    XtSetArg(blist[i],XtNfont,
             numberLineClassRec.numberLine_class.big_font); i++;
#ifdef XAW_BC
  nl->key_w =XtCreateManagedWidget("text",asciiStringWidgetClass,nlw,blist,i);
#else
  nl->key_w =XtCreateManagedWidget("text",asciiTextWidgetClass,nlw,blist,i);
#endif
  XtSetKeyboardFocus(nlw, nl->key_w);

  nl->old_hd_x = 0; nl->old_hd_y = 0;
  nl->head_val = nl->prec;
  nl->ptr_val = nl->prec;
  if (numberLineClassRec.numberLine_class.big_font)
    i = XTextWidth(numberLineClassRec.numberLine_class.big_font,
                   nl->title, strlen(nl->title)) + 50;
  else i = 10*strlen(nl->title) + 50;
  if (nlw->core.width == 0) nlw->core.width = (i < 225 ? 225 : i);
  if (nlw->core.height == 0) nlw->core.height = 110;


  XtAddEventHandler(nl->target_w, EnterWindowMask | PointerMotionMask |
                    ButtonPressMask | LeaveWindowMask,
                    False, TargetHandler,(caddr_t)nlw);
  XtAddEventHandler(nl->key_w, KeyPressMask, False, KeyHandler,(caddr_t)nlw);
} /* Initialize() */

static void Realize(widget, mask, attributes)
    Widget widget;
    Mask *mask;
    XSetWindowAttributes *attributes;
{
  NumberLinePart *nl = &((NumberLineWidget)widget)->numberLine;
  int z;

  (*superclass->core_class.realize)(widget, mask, attributes);

  z = 5;                        XtMoveWidget(nl->def_w,  z, 5);
  z += nl->def_w->core.width + 5;   XtMoveWidget(nl->prev_w, z, 5);
  z += nl->prev_w->core.width + 5;  XtMoveWidget(nl->key_w,  z, 5);

  XtRealizeWidget(nl->target_w);
  XDefineCursor(dpy, XtWindow(widget),
                numberLineClassRec.numberLine_class.pointer_cursor);
  XDefineCursor(dpy, XtWindow(nl->target_w),
                numberLineClassRec.numberLine_class.invisible_cursor);
} /* Realize() */

static void Destroy(w)
    Widget w;
{
  NumberLinePart *nl = &((NumberLineWidget)w)->numberLine;
  NumberLineCancelType(XtParent(w));
  XtRemoveEventHandler(nl->key_w,XtAllEvents,True,(XtEventHandler)NULL,
                       (caddr_t)NULL);
  XtFree((caddr_t) nl->string);
} /* Destroy() */

/*FORWARD*/static void set_pointer(), set_pointer_head(), unset_pointer_head();

/*ARGSUSED*/
static void Redisplay(w, event, region)
    Widget w;
    XEvent *event;
    Region region;
{
  Window  window = XtWindow(w);
  NumberLineWidget nlw = (NumberLineWidget) w;
  NumberLinePart *nl = & nlw->numberLine;
  CorePart *core = & ((NumberLineRec *) w)->core;
  XExposeEvent *xev = (XExposeEvent *) event;
  GC gc = numberLineClassRec.numberLine_class.gc;
  XFontStruct *smallFont = numberLineClassRec.numberLine_class.small_font;
  double order, temp; /* used in computing tics */
  double next, ticint;
  int x;
  int nextmandoc, lastmandoc;
  char s[20];

  if (xev->count) return; /* skip all regions except last (count == 0) */
  XClearWindow(dpy, window);
  /* Compute layout */
  nl->minpt = 20;
  nl->maxpt = nlw->core.width-20;
  nl->yline = /* (nlw->core.height-NL_HDR_HEIGHT)/2+ */ NL_HDR_HEIGHT +
        nl->def_w->core.height + 15;
  nl->ymens = nl->yline+2*nl->ticsize + /*smallfont->height ==*/ 10;
  if (nl->minpt >= nl->maxpt)
    return; /* give up, window too small (||| should make geometry request) */

  /* Configure subwindow target locations */
  XtConfigureWidget(nl->target_w, 0, nl->yline-20, core->width, 40, 0);
  XtConfigureWidget(nl->title_w, 5, nl->ymens+5, core->width-10,
                    nl->title_w->core.height, 0);
  (labelClassRec.core_class.expose)(nl->title_w, (XEvent *)0, (Region)0);
  /* Draw the numberLine */
  XDrawLine(dpy, window, gc, nl->minpt, nl->yline, nl->maxpt, nl->yline);
  XDrawLine(dpy, window, gc, nl->minpt, nl->yline-(int)(nl->ticsize*1.5),
            nl->minpt, nl->yline+(int)(nl->ticsize*1.5));
  XDrawLine(dpy, window, gc, nl->maxpt, nl->yline-(int)(nl->ticsize*1.5),
            nl->maxpt, nl->yline+(int)(nl->ticsize*1.5));

  /* print min numbering */
  if (nl->min != 0) sprintf(s, "%5f",nl->min);
  else sprintf(s,"   %1f   ",nl->min);
```

```
    XDrawImageString(dpy,window, gc, nl->minpt -
            (smallFont ? XTextWidth(smallFont,"000",3) : 18), nl->ymms, s, 5);
    nextnumloc = nl->minpt + (smallFont ? XTextWidth(smallFont,"00",2) : 9);
    sprintf(s, "%f      ",nl->max);
    XDrawImageString(dpy,window, gc, nl->maxpt -
            (smallFont ? XTextWidth(smallFont,"000",3) : 18), nl->ymms, s, 5);
    lastnumloc = nl->maxpt - 4*(smallFont ? XTextWidth(smallFont,"000",3):18);

    nl->sf = (nl->maxpt - nl->minpt)/(nl->max - nl->min);
    ticint = (nl->max - nl->min)/NL_MAX_TICS;
    /* determine tic interval based on a round number (10, 25, 50...) */
    order = pow(10.0, floor(log10(ticint)));
    temp = ticint/order;
    if (temp <= 1.0001) temp = 1.;
    else if (temp <= 2.5001) temp = 2.5;
    else if (temp <= 5.0001) temp = 5.0;
    else temp = 10.0;
    ticint = temp*order;

    for (next =  ticint * ceil(nl->min/ticint); next < nl->max;
        next += ticint) {
      x = (next-nl->min)*nl->sf+nl->minpt;
      XDrawLine(dpy, window, gc, x, nl->yline, x, nl->yline + nl->ticsize);
      if (smallFont) x -= XTextWidth(smallFont,"00",2); else x -= 9;
      if (x >= nextnumloc && x <= lastnumloc) {
          sprintf(s, "%f      ", next);
          XDrawImageString(dpy, window, gc, x, nl->ymms, s, 4);
          nextnumloc = x + (smallFont ? XTextWidth(smallFont,"00000",5) : 50);
      }
    }
    ticint *= 0.2;
    for (next = ticint*ceil(nl->min/ticint); next < nl->max;  next += ticint) {
      x = (next - nl->min)*nl->sf + nl->minpt;
      XDrawLine(dpy, window, gc, x, nl->yline,
                x, nl->yline + (int)(nl->ticsize*0.5));
    }

    nl->gran = ticint * 0.125;
    while (nl->gran * nl->sf < 1.0) nl->gran *= 2.0;

    nl->old_ptr_x = 0; /* so that old pointer won't be "erased" */
    set_pointer(nlw,gc,window,(int)((nl->ptr_val-nl->min) * nl->sf+nl->minpt),
                nl->yline-1, nl->ptr_val);
}

/*
 * a widget get_values_hook procedure to twiddle for retrieval of real
 * value from a "percent-ized" widget
 */
static void GetValuesHook (widget, args, argCount)
Widget widget;
ArgList args;
Cardinal *argCount;
{
    NumberLineWidget nlw = (NumberLineWidget)widget;
    int i = *argCount;
    double pres = nlw->numberLine.pres;

    if (nlw->numberLine.percentage)
      pres = nlw->numberLine.pres*nlw->numberLine.def/100.;
    while (i > 0)
      if (args[--i].name == XtNpresent)
        *((double *)args[i].value) = pres;
    }

    /*ARGSUSED*/
    static XtGeometryResult GeometryHandler(w, request, geometry_return)
        Widget w;
        XtWidgetGeometry *request;
        XtWidgetGeometry *geometry_return;
    {
      return(XtGeometryYes); /* always allow subwidgets to resize themselves */
    }

    static void NewVal(nlw, value)
    NumberLineWidget nlw;
    double value;
    {
      NumberLinePart *nl = & nlw->numberLine;
      int pt;

      nl->ptr_val =(value < nl->min ? nl->min :(value > nl->max ? nl->max :value));
      pt = (nl->ptr_val - nl->min)*nl->sf + nl->minpt;
      unset_pointer_head(nlw);
      set_pointer(nlw, numberLineClassRec.numberLine_class.gc,
                  XtWindow((Widget)nlw), pt, nl->yline-1, nl->ptr_val);
      nl->prev = nl->pres;
      nl->pres = nl->ptr_val;
    }

    /*ARGSUSED*/
    static void Notify(w, event, params, numParams)
        Widget w;
        XEvent *event;/*unused*/
        String *params;/*unused*/
        Cardinal *numParams;/*unused*/
    {
      XtCallCallbacks(w, XtNcallback, (caddr_t)NULL);
    }

    /*ARGSUSED*/
    static void KeyHandler(w, client_data, event)
        Widget w;
        caddr_t client_data;
        XKeyEvent *event;
    {
      NumberLineWidget nlw = (NumberLineWidget) client_data;
      char buf[64];
      int bytes;
      double value;

      bytes = XLookupString(event, buf, 2, (KeySym *)NULL, (XComposeStatus *)NULL);
      if (bytes >= 1 && (buf[0] == '\n' || buf[0] == '\r')) {
    #ifdef XAW_BC
        sscanf(nlw->numberLine.string,"%lf",&value);
    #else /* !XAW_BC */
        {
          Arg args[5];
          int i=0;
          char *widgetstring;

          XtSetArg(args[i], XtNstring, &widgetstring); i++;
```

```
    XtGetValues(nlw->numberLine.key_w, args, i);
    sscanf(widgetstring,"%lf",&value);
   }
#endif XAW_BC
   NewVal(nlw, value);
   Notify((Widget) nlw, (XEvent *)event, (String *)NULL, (Cardinal *)NULL);
  }
 }

 /*ARGSUSED*/
static void TargetHandler(w, client_data, event)
    Widget w;
    caddr_t client_data;
    XEvent *event;
{
 NumberLineWidget nlw = (NumberLineWidget) client_data;
 NumberLinePart *nl = & nlw->numberLine;
 XPointerMovedEvent *mevent = (XPointerMovedEvent *) event;

 switch(event->type) {
 case MotionNotify:
  {
   Window root, child;
   int x, y, win_x, win_y;
   unsigned int mask;
   if (XQueryPointer(dpy, mevent->window, &root, &child, &x, &y,
                     &win_x, &win_y, &mask)) {
       nl->head_val = (win_x - nl->minpt)/nl->sf + nl->min;
       /*give user--friendly granularity*/ /*wont rint() but not on Cray*/
       nl->head_val = nl->gran*floor(0.5+nl->head_val/nl->gran);
       nl->head_val = nl->head_val < nl->min ? nl->min :
         (nl->head_val > nl->max ? nl->max : nl->head_val);
       set_pointer_head(nlw,(int)((nl->head_val-nl->min)*nl->sf + nl->minpt),
                        nl->yline-2);
       return;
   }
   break;
  }
 case ButtonPress:
  if (nl->ptr_val != nl->head_val) {
   NewVal(nlw, nl->head_val);
   Notify((Widget)nlw, event, (String *)NULL, (Cardinal *)NULL);
  }
  break;
 case LeaveNotify:
  unset_pointer_head(nlw);
  return;
 default:
  return;
 }
}

/*ARGSUSED*/
static void DefaultCallback(widget, client_data, call_data)
    Widget widget;
    caddr_t client_data, call_data;
{
 NumberLineWidget nlw = (NumberLineWidget) client_data;

 if (nlw->numberLine.percentage)
  NewVal(nlw, (double)100.);
 else
  NewVal(nlw, nlw->numberLine.def);
 Notify((Widget)nlw, (XEvent *)call_data, (String *)NULL, (Cardinal *)NULL);
}

/*ARGSUSED*/
static void PreviousCallback(widget, client_data, call_data)
    Widget widget;
    caddr_t client_data, call_data;
{
 NumberLineWidget nlw = (NumberLineWidget) client_data;
 NewVal(nlw, nlw->numberLine.prev);
 Notify((Widget)nlw, (XEvent *)call_data, (String *)NULL, (Cardinal *)NULL);
}

static void set_pointer(nlw, gc, w, x, y, val)
    NumberLineWidget nlw;
    GC gc;
    Window w;
    int x, y;
    double val;
{
 NumberLinePart *nl = & nlw->numberLine;
 static char s[40];
#ifdef XAW_BC
 static XtTextBlock textblock = {0, 10, s};
#endif /* XAW_BC */
 XPoint points[4];

 if (nl->old_ptr_x)
   XClearArea(dpy, w, 0, y-11-10-15, nlw->core.width, 15+10+11+1, False);
 XSetLineAttributes(dpy, gc, 2, LineSolid, CapButt, JoinMiter);
 points[0].x = x-5; points[0].y = y-5;
 points[1].x = x; points[1].y = y;
 points[2].x = x+5; points[2].y = y-5;
 points[3].x = x-5; points[3].y = y-5;
 XFillPolygon(dpy, w, gc, points, 3, Convex, CoordModeOrigin);
 XDrawLine(dpy, w, gc, x, y-5, x, y-15);
 XSetLineAttributes(dpy, gc, 0, LineSolid, CapButt, JoinMiter);
 XDrawLine(dpy, w, gc, nl->minpt, (int) (nl->yline-(nl->ticsize*1.5)),
           nl->minpt, (int) (nl->yline+(nl->ticsize * 1.5)));
 XDrawLine(dpy, w, gc, nl->maxpt, (int) (nl->yline-(nl->ticsize*1.5)),
           nl->maxpt, (int) (nl->yline+(nl->ticsize * 1.5)));
 sprintf(s,"%7.3f",val);
 XDrawImageString(dpy, w, gc, x-(3*6), y-11-10,
                  s, strlen(s));
#ifdef XAW_BC
 textblock.length = strlen(s);
 XtTextSetLastPos(nl->key_w, XtTextTopPosition(nl->key_w));
 XtTextSetInsertionPoint(nl->key_w, (XtTextPosition)0);
 XtTextSetSelection(nl->key_w, (XtTextPosition)0, (XtTextPosition)0);
 XtTextReplace(nl->key_w, (XtTextPosition)0, (XtTextPosition)0, &textblock);
 XtTextSetInsertionPoint(nl->key_w, (XtTextPosition)0);
#else /* ! XAW_BC */
 {
  Arg args[5];
  int i=0;
  char *news = XtNewString(s);
  XtSetArg(args[i], XtNstring, news); i++;
  XtSetValues(nl->key_w, args, i);
  XtFree(nl->string);
```

```
    nl->string = news;
    XawTextSetInsertionPoint(nl->key_w, (XawTextPosition)0);
  }
#endif /* XAW_BC */

  nl->old_ptr_x = x; nl->old_ptr_y = y;
}

static void set_pointer_head(nlw, x, y)
    NumberLineWidget nlw;
    int x, y;
{
 NumberLinePart *nl = &nlw->numberLine;
 Window w = XtWindow(nlw);
 GC gc = numberLineClassRec.numberLine_class.gc;

 XSetFunction(dpy, gc, GXinvert);
 XSetPlaneMask(dpy, gc, 1);
 if (nl->old_hd_x) {
  XDrawLine(dpy, w, gc, nl->old_hd_x, nl->old_hd_y, nl->old_hd_x-5,
            nl->old_hd_y-5);
  XDrawLine(dpy, w, gc, nl->old_hd_x, nl->old_hd_y, nl->old_hd_x+5,
            nl->old_hd_y-5);
  XDrawPoint(dpy, w, gc, nl->old_hd_x, nl->old_hd_y);
 }
 XDrawLine(dpy, w, gc, x, y, x-5, y-5);
 XDrawLine(dpy, w, gc, x, y, x+5, y-5);
 XDrawPoint(dpy, w, gc, x, y);
 nl->old_hd_x = x; nl->old_hd_y = y;
 XSetFunction(dpy, gc, GXcopy);
 XSetPlaneMask(dpy, gc, AllPlanes);
}

static void unset_pointer_head(nlw)
    NumberLineWidget nlw;
{
 NumberLinePart *nl = &nlw->numberLine;
 GC gc = numberLineClassRec.numberLine_class.gc;
 XSetFunction(dpy, gc, GXinvert);
 XSetPlaneMask(dpy, gc, 1);
 if (nl->old_hd_x) {
  XDrawLine(dpy, XtWindow(nlw), gc, nl->old_hd_x, nl->old_hd_y,
            nl->old_hd_x-5, nl->old_hd_y-5);
  XDrawLine(dpy, XtWindow(nlw), gc, nl->old_hd_x, nl->old_hd_y,
            nl->old_hd_x+5, nl->old_hd_y-5);
  XDrawPoint(dpy, XtWindow(nlw), gc, nl->old_hd_x, nl->old_hd_y);
 }
 nl->old_hd_x = 0; nl->old_hd_y = 0;
 XSetFunction(dpy, gc, GXcopy);
 XSetPlaneMask(dpy, gc, AllPlanes);
}


/**********************************************************
 *
 * Public Procedures
 *
 **********************************************************/

/*
 *
 * The following keeps track of which number lines are present since
 * duplicates would have non-intuitive behavior and are hereby disallowed.
 *
 */


static Widget nlPresent[10][10] = {(Widget)0, };
static XContext erContext = NULL;
typedef struct {
int place, type;
} erRec;

/*PUBLIC*/ Boolean
NumberLineEnroll(w, place, type) /* returns true if successful */
    Widget w;
    int place, type;
{
 erRec *rec;
 if (!erContext) erContext = XUniqueContext();
 rec = (erRec *) XtMalloc(sizeof(erRec));
 rec->place = place; rec->type = type;
 XSaveContext(dpy, XtWindow(w), erContext, (caddr_t) rec);
 if (nlPresent[place][type])
  return (False);
 else {
  nlPresent[place][type] = w;
  return (True);
 }
}

/*PUBLIC*/ void
NumberLineCancelType(w)
    Widget w;
{
 erRec *rec;
 if (XFindContext(dpy, XtWindow(w), erContext, (caddr_t *)&rec))
  return; /* error getting context */
 nlPresent[rec->place][rec->type] = (Widget) 0;
 XDeleteContext(dpy, XtWindow(w), erContext);
 XtFree((caddr_t) rec);
}

/*PUBLIC*/ Boolean
NumberLineCheckType(place, type)
int place, type;
{
 if (nlPresent[place][type]) return(True);
 else return(False);
}

/*PUBLIC*/ void
NumberLineResetAll()
{
 int i, j;
 ShellWidget s;

 for (i = 0; i < 10; i++)
  for (j = 0; j < 10; j++)
   if (nlPresent[i][j]) {
       s = (ShellWidget) nlPresent[i][j];
       DeferCallback(nlPresent[i][j], (caddr_t) *(s->composite.children),
```

```
                    (caddr_t)0);
    }
}

/*PUBLIC*/ void
NumberLineDestroyAll()
{
 int i, j;
 Widget w;

 for (i = 0; i < 10; i++)
  for (j = 0; j < 10; j++)
   if (wasIPressed[j][i]) {
       NumberLineCancelType(w);
       XtDestroyWidget(w);
   }
}
/*
 *
 * Get current value
 *
 */

/*PUBLIC*/ void NumberLineGetValue (w, value, done)
Widget w;
double *value; /* RETURN */
int *done;     /* RETURN */
{
 NumberLinePart *nl = &((NumberLineWidget) w)->numberLine;
 if (nl->percentage)
  *value = nl->pres*nl->def/100.;
 else
  *value = nl->pres;
 *done = nl->done;
}
```

nlTest.c

```
#ifndef lint
static char *vcs="$Header: /usr/local/src/nl/RCS/nlTest.c,v 2.1 90/08/10 16:15:32 tldavis Exp $";
#endif  lint
/*
 * Test NumberLine X11 Version, Copyright 1988 by Timothy L. Davis
 * $Source: /usr/lib/src/nl/RCS/nlTest.c,v $
 */

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include "NumberLine.h"

Display *dpy;
Window root;
int screen;

/*ARGSUSED*/
static void callback(widget, client_data, call_data)
Widget widget;
caddr_t client_data, call_data;
{
 static double value;
 static int done;

 NumberLineGetValue(widget, &value, &done);
 printf("Number Line Callback:   value = %lf\n", value);
 if (done) {
  printf("Done!\n");
  exit(0);
 }
}
main(argc, argv)
    int argc;
    char **argv;
{
 Widget shell;
 Widget nlw;
 double min = 1.0 , max = 10.0 , present = 2.0 , def = 5.0;
 Arg args[10];
 int i;

 printf("XtInitialize...\n");
 shell = XtInitialize(argv[0],"Testing", (XrmOptionDescRec *)NULL,
                      (Cardinal)0, (Cardinal *)&argc, (String *)argv);
 while (argc && --argc)
  printf("unuse. argument:   %s\n",argv[argc]);

 dpy = XtDisplay(shell);
 screen = DefaultScreen(dpy);
 root = RootWindow(dpy, screen);
 XSynchronize(dpy, 1); /* makes it slow but easier for bug tracking */

 printf("XtCreateManaged...dpy %x,  screen %d, root %d\n",dpy,screen,root);
 i=0;
 XtSetArg(args[i], XtNminimum, &min);i++;
 XtSetArg(args[i], XtNmaximum, &max);i++;
 XtSetArg(args[i], XtNpresent, &present);i++;
 XtSetArg(args[i], XtNdefault, &def);i++;
 XtSetArg(args[i], XtNprevious, &present);i++;
 nlw = XtCreateManagedWidget("numberLine", numberLineWidgetClass, shell,
                             args, i);

 XtAddCallback(nlw, XtNcallback, callback, nlw);

 printf("XtRealizeWidget(shell)...\n");
 XtRealizeWidget(shell);
 {
  XEvent event;
  for (;;) {
   XNextEvent(dpy, &event);
   XtDispatchEvent(&event);
  }
 }
}
```

C.5 Plot widget

These files define the Plot widget, which dynamically displays data on an X-vs Y1,Y2,Y3 or strip-chart plots, with options for oscilloscope or moving-strip-chart mode and automatic bound checking and rescaling. Pointer motion over the window

displays cross-haired coordinate values, and clicking places a printout of those values on the plot. See `Plot.h` for a list of resources available for interfacing with user programs.

Plot.h

```
#ifndef _Plot_h
#define _Plot_h
/*****************************************************************************
 *
 * Strip Chart Widget
 *
 *****************************************************************************/

#ifdef XAW_BC
#define XtNlabel                "label"
#define XtNfont                 "font"
#endif XAW_BC
#define XtNpaperSpeed           "paperSpeed"
#define XtNdecayTime            "decayTime"
#define XtNplaceX               "placeX"
#define XtNplace1               "place1"
#define XtNplace2               "place2"
#define XtNplace3               "place3"
#define XtNtypeX                "typeX"
#define XtNtype1                "type1"
#define XtNtype2                "type2"
#define XtNtype3                "type3"
#define XtNminX                 "minX"
#define XtNmin1                 "min1"
#define XtNmin2                 "min2"
#define XtNmin3                 "min3"
#define XtNmaxX                 "maxX"
#define XtNmax1                 "max1"
#define XtNmax2                 "max2"
#define XtNmax3                 "max3"
#define XtNlabelX               "labelX"
#define XtNlabel1               "label1"
#define XtNlabel2               "label2"
#define XtNlabel3               "label3"
#define XtNunitX                "unitX"
#define XtNunit1                "unit1"
#define XtNunit2                "unit2"
#define XtNunit3                "unit3"
#define XtNautoScaleX           "autoScaleX"
#define XtNautoScale1           "autoScale1"
#define XtNautoScale2           "autoScale2"
#define XtNautoScale3           "autoScale3"
#define XtNstripMode            "stripMode"
#define XtNplotback             "plotback"
#define XtNcolor1               "color1"
#define XtNcolor2               "color2"
#define XtNcolor3               "color3"
#define XtNtiming               "timing"

extern WidgetClass plotWidgetClass;

extern void PlotData(); /* w, pressures */
   /* Widget w; */
   /* double *pressures; */
extern void PlotModifySimulation(); /* w, place, type, value */
   /* Widget w; */
   /* int place, type; */
   /* double value; */

#endif _Plot_h
```

PlotP.h

```
#ifndef _PlotP_h
#define _PlotP_h

#include <X11/CompositeP.h>
#include "Plot.h"

/*****************************************************************************
 *
 *  Plot Widget Class Definitions
 *
 *****************************************************************************/

#define XtRDouble               "double"

typedef struct {
  GC        gc1, gc2, gc3, pointerGC;
  XFontStruct *small_font;
  Cursor    pointer_cursor, plotwin_cursor;
} PlotClassPart;

typedef struct {
  CoreClassPart      core_class;
  CompositeClassPart composite_class;
  PlotClassPart      plot_class;
} PlotClassRec;

extern PlotClassRec plotClassRec;

/*****************************************************************************
 *
 *  Plot Widget Definitions
 *
 *****************************************************************************/


typedef struct {
  /* accessible from resources */
  Pixel color1, color2, color3, foreground, background, plotback;
    /* Colors of three plot lines, general background and plot bkground */
  char *title;          /* title of entire plot (unused at present) */
  double paperSpeed;    /* for stripchart: in cm/sec on screen */
  double decayTime; /* length of time on X-Y plot pixel stays before erasure */
  int placeX, place1, place2, place3; /* variables place codes for all axes */
  int typeX, type1, type2, type3;     /* variable type codes */
  char *labelX, *label1, *label2, *label3; /* axis names */
  char *unitX, *unit1, *unit2, *unit3;
  double minX, min1, min2, min3; /* all minimum values */
  double maxX, max1, max2, max3; /* all max values */
  Boolean autoX, auto1, auto2, auto3; /* autoscalable flags */
  Boolean stripMode; /* True = scrolling stripchart, False = oscilliscope */
  unsigned int timing;/* =0: continuous plot, = other codes: 1 pt of cycle */
```

```
    /* private state */
    XtCallbackList callbacks;
    int plotWidth, plotHeight; /* inner plot size, calculated */
    int titlePad, labelPad; /* amount of space around top and bottom */
    int xoff; /* x-axis position for pointer plot position string. */
    double currentTime;
    double xf1, xf2, xf3;     /* all computed scale factors data->pixel */
    int slideDots; /* number of pixels per slide operation in stripchart */
    int nextHash; /* next place to get a time tic in stripchart (time) mode */
    int lastPtX, lastPt1, lastPt2, lastPt3; /* last pixels plotted */
    int phase2, phase3; /* Dash phase positions. */
    int outX, out1, out2, out3;
    double lastX, last1, last2, last3;
    double sumX, sum1, sum2, sum3;
    unsigned int cntX, cnt1, cnt2, cnt3;
    Window plotwin;/* inner window where plotting is going on. */
    Drawable plotpix; /* background pixmap of plotwin, contains actual plot */
    Widget label1w, label2w, label3w; /* Label widgets */
    Widget refreshButton, modButton; /* active command buttons */
} PlotPart;

typedef struct {
  CorePart       core;
  CompositePart compositn;
  PlotPart       plot;
} PlotRec, *PlotWidget;

#endif _PlotP_h
```

Plot.c

```
#ifndef lint
static char *vcs="$Header: /usr/hat/src/plot/RCS/Plot.c,v 3.1 90/08/28 17:31:31 tldavis Exp $";
#endif lint

/*****************************************************************************
 *
 * Plot widget for X11 Cardiovascular Simulator
 * Copyright 1988 by Timothy L. Davis
 *
 * $Source: /usr/hat/src/plot/RCS/Plot.c,v $
 * $Log:        Plot.c,v $
 * Revision 3.1  90/08/28  17:31:31  tldavis
 * Removed a bug that cropped up with an absent X resource file.
 *
 * Revision 3.0  90/08/28  14:41:51  tldavis
 * Added support for the XtNtiming set of flags (at least 16 bit int).
 * XtNtiming has flags which determine the current portion of cardiac cycle,
 * the portion to plot, and whether to average data before plotting.
 *
 * Revision 1.1  90/08/19  18:42:25  tldavis
 * Initial revision
 *
 * Revision 2.2  90/08/18  18:36:40  tldavis
 * Many changes:  reworked all the display code for speed and bugfixes.
 * Now fully supports color resources.
 *
 * Revision 2.1  90/08/10  10:57:23  dat
 * moved Xaw header files to X11/Xaw for R4
 *
 * Revision 2.0  90/08/06  17:59:46  tldavis
 * Final X11R3 version.
 *
 * Revision 1.24  90/08/06  17:05:47  tldavis
 * Fixed problems with color on Suns.
 *
 * Revision 1.23  90/05/31  16:29:06  tldavis
 * Pre-ACIS checkin.
 *
 * Revision 1.22  90/02/09  15:28:45  tldavis
 * Avoids redisplaying at almost all costs.
 * Tweaked to keep plotting in some place after exposures.
 *
 * Revision 1.21  89/10/12  22:15:29  tldavis
 * Added possibility for bitmap saving, if you #define PIXDRAW.
 *
 * Revision 1.20  89/10/10  23:13:12  tldavis
 * mwm support, works well but no saving of background pixmap.
 * Works with either stripchart or scope or xy mode.
 * R3 fonts fixed.
 * Revision 1.13  89/02/22  12:27:07  tldavis
 * X11R3 version (no saves of old data).
 * Revision 1.10  88/10/10  21:45:34  hat
 * Fixed several bugs to do with labels, removed ball for speed reasons.
 * Revision 1.2  88/08/03  18:21:34  tldavis
 * First successful compile of X11 version.
 * Revision 1.1  88/08/03  14:53:09  tldavis
 * Initial revision
 *
 * PLOT: Everything needed for the plot window, both stripchart and X v. Y
 * style plots.  Accepts up to 3 separate variables on the Y axis, plotted
 * against a common X axis.  Each variable may be "auto-rescaling", meaning
 * that the plot axes for that variable are recalculated if the values go
 * out of bounds.
 *
 * WidgetClass is plotWidgetClass
 *
 * void PlotData(Window w, double *pressures) is the main routine to
 *    provide the plot widget with a new set of data values to be plotted.
 *    pressures should point to a 9-element array containing the 6 pressures,
 *    time, Cl(t) and Cr(t) for that time, as produced inside simulate.c
 *    Generally, this routine should only be called by the control loop which
 *    polls the simulation data pipe and broadcasts new data to all active
 *    plots.
 *
 * void PlotModifySimulation(Window w, int place, int type, double value)
 *    allows for changes to the widget's view of the simulation, so that
 *    variables for plotting may be calculated appropriately from the given
 *    pressures.
 *
 * define PIXDRAW to draw all data to the background pixmap.  It won't be lost
 *    when the window is obscured, but this is MUCH slower than the other way.
 *
 * define MOVINGBALL to put a pen-ball head at the right of the stripchart.
 *    This also slows down the animation.
 *
 * define SAVEDATA to save a list of previously-plotted data.
 *
 *****************************************************************************/
#ifdef NOTDEF
#define PIXDRAW /* Save plot data in background pixmaps SURE IS SLOW! */
#endif NOTDEF
```

```
#include <stdio.h>
#include <strings.h>
#include <X11/IntrinsicP.h>
#include <X11/StringDefs.h>
#include <X11/Xaw/Command.h>
#include <X11/cursorfont.h>
#include "PlotP.h"
#include "../form/PlotForm.h" /* For Modify Axes menu. */
#include "../sim/CVDefs.h"
#include <math.h>
extern double getval(); /* from ../sim/eqns.c */
extern unsigned int UpdatePeriod(); /* from ../sim/eqns.c */
extern double drawAxis();

extern Display *dpy;
extern Window root;
extern int screen;

#define PAD 40 /* pad around outside of graph (to include labels/menu) */
#define AXISPAD 12 /* pad for axis dimensions */
#define SAMPLEPAD 32 /* pad for sample line dashes */
/* screen resolution in pixels per cm */
#define RESOLUTION (10.*XDisplayWidth(dpy,XDefaultScreen(dpy))/ \
                    XDisplayWidthMM(dpy,XDefaultScreen(dpy)))

#define offset(field) XtOffset(PlotWidget,plot.field)
static XtResource resources[] =
  {
    {XtNcolor1, XtCForeground, XtRPixel, sizeof(Pixel),
      offset(color1), XtRString, "XtDefaultForeground"},
    {XtNcolor2, XtCForeground, XtRPixel, sizeof(Pixel),
      offset(color2), XtRString, "XtDefaultForeground"},
    {XtNcolor3, XtCForeground, XtRPixel, sizeof(Pixel),
      offset(color3), XtRString, "XtDefaultForeground"},
    {XtNforeground, XtCForeground, XtRPixel, sizeof(Pixel),
      offset(foreground), XtRString, "XtDefaultForeground"},
    {XtNbackground, XtCBackground, XtRPixel, sizeof(Pixel),
      offset(background), XtRString, "XtDefaultBackground"},
    {XtNplotback, XtCBackground, XtRPixel, sizeof(Pixel),
      offset(plotback), XtRString, "XtDefaultBackground"},
    {XtNlabel, XtCLabel, XtRString, sizeof(String),
      offset(title), XtRString, NULL},
    {XtNpropSpeed, XtCValue, XtRDouble, sizeof(double),
      offset(propSpeed), XtRString, "5.0"},
    {XtNdecayTime, XtCValue, XtRDouble, sizeof(double),
      offset(decayTime), XtRString, "1.0"},
    {XtNplaceX, XtCValue, XtRInt, sizeof(int),
      offset(placeX), XtRString, "0"},
    {XtNplace1, XtCValue, XtRInt, sizeof(int),
      offset(place1), XtRString, "-1"},
    {XtNplace2, XtCValue, XtRInt, sizeof(int),
      offset(place2), XtRString, "-1"},
    {XtNplace3, XtCValue, XtRInt, sizeof(int),
      offset(place3), XtRString, "-1"},
    {XtNtypeX, XtCValue, XtRInt, sizeof(int),
      offset(typeX), XtRString, "0"},
    {XtNtype1, XtCValue, XtRInt, sizeof(int),
      offset(type1), XtRString, "-1"},
    {XtNtype2, XtCValue, XtRInt, sizeof(int),
      offset(type2), XtRString, "-1"},
    {XtNtype3, XtCValue, XtRInt, sizeof(int),
      offset(type3), XtRString, "-1"},
    {XtNlabelX, XtCValue, XtRString, sizeof(String),
      offset(labelX), XtRString, "Time"},
    {XtNlabel1, XtCValue, XtRString, sizeof(String),
      offset(label1), XtRString, NULL},
    {XtNlabel2, XtCValue, XtRString, sizeof(String),
      offset(label2), XtRString, NULL},
    {XtNlabel3, XtCValue, XtRString, sizeof(String),
      offset(label3), XtRString, NULL},
    {XtNminX, XtCValue, XtRDouble, sizeof(double),
      offset(minX), XtRString, "0."},
    {XtNmin1, XtCValue, XtRDouble, sizeof(double),
      offset(min1), XtRString, "0."},
    {XtNmin2, XtCValue, XtRDouble, sizeof(double),
      offset(min2), XtRString, "0."},
    {XtNmin3, XtCValue, XtRDouble, sizeof(double),
      offset(min3), XtRString, "0."},
    {XtNmaxX, XtCValue, XtRDouble, sizeof(double),
      offset(maxX), XtRString, "100."},
    {XtNmax1, XtCValue, XtRDouble, sizeof(double),
      offset(max1), XtRString, "100."},
    {XtNmax2, XtCValue, XtRDouble, sizeof(double),
      offset(max2), XtRString, "100."},
    {XtNmax3, XtCValue, XtRDouble, sizeof(double),
      offset(max3), XtRString, "100."},
    {XtNunitX, XtCValue, XtRString, sizeof(String),
      offset(unitX), XtRString, NULL},
    {XtNunit1, XtCValue, XtRString, sizeof(String),
      offset(unit1), XtRString, NULL},
    {XtNunit2, XtCValue, XtRString, sizeof(String),
      offset(unit2), XtRString, NULL},
    {XtNunit3, XtCValue, XtRString, sizeof(String),
      offset(unit3), XtRString, NULL},
    {XtNautoScaleX, XtCValue, XtRBoolean, sizeof(Boolean),
      offset(autoX), XtRBoolean, False},
    {XtNautoScale1, XtCValue, XtRBoolean, sizeof(Boolean),
      offset(auto1), XtRBoolean, False},
    {XtNautoScale2, XtCValue, XtRBoolean, sizeof(Boolean),
      offset(auto2), XtRBoolean, False},
    {XtNautoScale3, XtCValue, XtRBoolean, sizeof(Boolean),
      offset(auto3), XtRBoolean, False},
    {XtNstriphfeels, XtCValue, XtRBoolean, sizeof(Boolean),
      offset(striphfeels), XtRBoolean, False},
    {XtNcallback, XtCCallback, XtRCallback, sizeof(caddr_t),
      offset(callbacks), XtRCallback, NULL},
    {XtNtiming, XtCValue, XtRInt, sizeof(int),
      offset(timing), XtRString, "0"},
  };
#undef offset


/*
 * FORWARD defines for the widget activity routines to be put in core struct
 * These routines follow widget interaction rules.
 */
static void ClassInitialize();
static void Initialize();
static void Realize();
static void Destroy();
static void Redisplay();
static void UpdateCalipers(), WriteCalipers(), EraseCalipers();
```

```
static Boolean SetValues();
static XtGeometryResult GeometryManager();

PlotClassRec plotClassRec = {
  { /* core_class fields */
#define superclass ((WidgetClass) &compositeClassRec)
    /* superclass               */    superclass,
    /* class_name               */    "Plot",
    /* widget_size              */    sizeof(PlotRec),
    /* class_initialize         */    ClassInitialize,
    /* class_part_initialize    */    NULL,
    /* class_inited             */    False,
    /* initialize               */    Initialize,
    /* initialize_hook          */    NULL,
    /* realize                  */    Realize,
    /* actions                  */    NULL, /* TOO SLOW */
    /* num_actions              */    0,
    /* resources                */    resources,
    /* num_resources            */    XtNumber(resources),
    /* xrm_class                */    NULLQUARK,
    /* compress_motion          */    True,
    /* compress_exposure        */    True,
    /* compress_enterleave      */    True,
    /* visible_interest         */    False,
    /* destroy                  */    Destroy,
    /* resize                   */    XtInheritResize,
    /* expose                   */    Redisplay,
    /* set_values               */    SetValues,
    /* set_values_hook          */    NULL,
    /* set_values_almost        */    XtInheritSetValuesAlmost,
    /* get_values_hook          */    NULL,
    /* accept_focus             */    NULL,
    /* version                  */    XtVersion,
    /* callback_private         */    NULL,
    /* tm_table                 */    NULL, /* TOO SLOW */
    /* query_geometry           */    XtInheritQueryGeometry,
    /* ***NEWER*** display_accel*/    XtInheritDisplayAccelerator,
    /* ***NEWER*** extension    */    NULL
  },
  { /* composite_class fields */
    /* geometry_manager */    GeometryManager,
    /* change_managed */      XtInheritChangeManaged,
    /* insert_child */        XtInheritInsertChild,
    /* delete_child */        XtInheritDeleteChild,
    /* ***NEWER*** extension */ NULL
  },
  { /* Plot fields */
    /* gc's */                (GC)NULL, (GC)NULL, (GC)NULL, (GC)NULL,
    /* small_font */          (XFontStruct *)NULL,
    /* pointer_cursor */      (Cursor)NULL, (Cursor)NULL,
  }
};
WidgetClass plotWidgetClass = (WidgetClass) &plotClassRec;

/**********************************************************************
 *
 * Private Procedures
 *
 **********************************************************************/
/*FORWARD*/ static void ModCallback(), RefreshCallback();

/*ARGSUSED*/
static void CvtStringToDouble(args, num_args, fromVal, toVal)
XrmValue *args;
Cardinal *num_args;
XrmValue *fromVal, *toVal;
{
 static double i;
 if (*num_args != 0)
  XtWarning("String to Double conversion needs no extra arguments");
 if (sscanf ((char *)fromVal->addr,"%lf",&i) == 1) {
  toVal->size = sizeof(double);
  toVal->addr = (caddr_t) &i;
 }
}

/*
 * registered private widget initialization and event handling routines:
 *
 */
#ifdef PIXDRAW
static GC whiteGC = (GC)0;
#endif PIXDRAW
static char Dash2[] = {1, 3, 0};
static char Dash3[] = {3, 5, 0};
static char *trans = "<Btn0p>:    notify() unset() unhighlight() \n";
static XtTranslations parsed_translations;

static void ClassInitialize()
{
 XGCValues gcvalues;
 unsigned long fontmask = 0;

 if (plotClassRec.plot_class.small_font =
     XLoadQueryFont(dpy,"*-courier-medium-r-*-*-*-80-*"))
  fontmask = GCFont; /* successful font acquisition, else default. */
 plotClassRec.plot_class.pointer_cursor = XCreateFontCursor(dpy, XC_arrow);
 if (plotClassRec.plot_class.small_font)
  gcvalues.font = plotClassRec.plot_class.small_font->fid;
 plotClassRec.plot_class.plotwin_cursor =
  XCreateFontCursor(dpy, XC_crosshair);

 gcvalues.graphics_exposures = False;
 plotClassRec.plot_class.gc1 = XCreateGC(dpy, root, GCGraphicsExposures |
                                         fontmask, &gcvalues);
 gcvalues.line_style = LineOnOffDash;
 plotClassRec.plot_class.gc2 = XCreateGC(dpy, root, GCGraphicsExposures |
                                         fontmask | GCLineStyle,
                                         &gcvalues);
 XSetDashes(dpy, plotClassRec.plot_class.gc2, 0, Dash2, sizeof(Dash2));
 gcvalues.line_style = LineOnOffDash;
 plotClassRec.plot_class.gc3 = XCreateGC(dpy, root, GCGraphicsExposures |
                                         fontmask | GCLineStyle,
                                         &gcvalues);
 XSetDashes(dpy, plotClassRec.plot_class.gc3, 0, Dash3, sizeof(Dash3));
 XtAddConverter(XtRString, XtRDouble, CvtStringToDouble, (ArgList)NULL, 0);

#ifdef PIXDRAW
 gcvalues.foreground = BlackPixel(dpy,screen);
 gcvalues.background = WhitePixel(dpy,screen);
 whiteGC = XCreateGC(dpy, root, GCForeground | GCBackground, &gcvalues);
#endif PIXDRAW
```

```
    gcvalues.function = GXcopy;
    plotClassRec.plot_class.pointerGC =
      XCreateGC(dpy, root, GCFunction | fourmask, &gcvalues);

    parsed translations = XtParseTranslationTable(trans);
} /* ClassInitialize() */


static void Initialize(request, new)
    Widget request, new;
{
  PlotWidget pw = (PlotWidget) new;
  PlotPart *plot = & pw->plot;
  static PlotRec blank = {{0,}, {0,},
                          {0, 0, 0, 0, 0, 0, 0,
                          0, 0.0, 0.0, -1, -1, -1, -1, -1, -1, -1, -1, 0, }};
  int i;
  Arg arg[10];

  XSetForeground(dpy, plotClassRec.plot_class.gc1, plot->color1);
  XSetBackground(dpy, plotClassRec.plot_class.gc1, plot->plotback);
  XSetForeground(dpy, plotClassRec.plot_class.gc2, plot->color2);
  XSetBackground(dpy, plotClassRec.plot_class.gc2, plot->plotback);
  XSetForeground(dpy, plotClassRec.plot_class.gc3, plot->color3);
  XSetBackground(dpy, plotClassRec.plot_class.gc3, plot->plotback);
  XSetForeground(dpy, plotClassRec.plot_class.pointerGC, plot->foreground);
  XSetBackground(dpy, plotClassRec.plot_class.pointerGC, plot->background);

  plot->titlePad = 5;
  plot->labelPad = 5;
  plot->slideData = 1;
  plot->outX = plot->out1 = plot->out2 = plot->out3 = 0;
  plot->phase2 = plot->phase3 = 0;
  plot->lastX = plot->last1 = plot->last2 = plot->last3 = 0.;
  plot->dX = 0.0;
  if (pw->core.width == 0) pw->core.width = 400;
  if (pw->core.height == 0) pw->core.height = 200;
  plot->plotwin = 0;

  i=0;
  XtSetArg(arg[i], XtNlabel, "Modify Axes");i++;
  plot->modButton =
    XtCreateManagedWidget("Modify Axes", commandWidgetClass, pw, arg, i);
  XtAddCallback(plot->modButton, XtNcallback, ModCallback, (caddr_t)pw);
  XtOverrideTranslations(plot->modButton, parsed translations);

  i=0;
  XtSetArg(arg[i], XtNlabel, "Refresh");i++;
  plot->refreshButton =
    XtCreateManagedWidget("Refresh", commandWidgetClass, pw, arg, i);
  XtAddCallback(plot->refreshButton, XtNcallback, RefreshCallback,(caddr_t)pw);

  /* Leave them unmanaged for now... manage them selectively in SetValues */
  i=0;
  XtSetArg(arg[i], XtNforeground, plot->color1);i++;
  XtSetArg(arg[i], XtNbackground, plot->background);i++;
  plot->label1w = XtCreateWidget("label1", labelWidgetClass, pw, arg, i);
  i=0;
  XtSetArg(arg[i], XtNforeground, plot->color2);i++;
  XtSetArg(arg[i], XtNbackground, plot->background);i++;
  plot->label2w = XtCreateWidget("label2", labelWidgetClass, pw, arg, i);
  i=0;
  XtSetArg(arg[i], XtNforeground, plot->color3);i++;
  XtSetArg(arg[i], XtNbackground, plot->background);i++;
  plot->label3w = XtCreateWidget("label3", labelWidgetClass, pw, arg, i);

  XtAddEventHandler(new, PointerMotionMask, False,UpdateCaliper, (caddr_t)0);
  XtAddEventHandler(new, LeaveWindowMask, False, EraseCaliper, (caddr_t)0);
  XtAddEventHandler(new, ButtonPressMask, False, WriteCaliper, (caddr_t)0);
  SetValues((Widget) &blank, request, new);
} /* Initialize() */


static void Realize(widget, mask, attributes)
    Widget widget;
    Mask *mask;
    XSetWindowAttributes *attributes;
{
  PlotPart *plot = &((PlotWidget)widget)->plot;

  *mask = *mask | CWBackingStore | CWSaveUnder;
  attributes->backing_store = Always;
  attributes->save_under = True;
  (*superclass->core_class.realize)(widget, mask, attributes);
  XDefineCursor(dpy, XtWindow(widget), plotClassRec.plot_class.pointer_cursor);
  plot->plotwin = XCreateSimpleWindow(dpy, XtWindow(widget), 1, 1, 1, 1, 0,
                                      (unsigned long)plot->foreground,
                                      (unsigned long)plot->plotback);
  XDefineCursor(dpy, plot->plotwin, plotClassRec.plot_class.plotwin_cursor);
  plot->plotpix = (Drawable)0;
  XMapWindow(dpy, plot->plotwin);
} /* Realize() */

static void Destroy(w)
    Widget w;
{
  PlotPart *plot = &((PlotWidget)w)->plot;
  #ifdef PIXDRAW
  /*|||888 XFreePixmap(dpy, (Pixmap)plot->plotpix); */
  #endif PIXDRAW
  XDestroyWindow(dpy, plot->plotwin);
}

/*
 *
 * Repaint the widget window
 *
 */
/*ARGSUSED*/
static void Redisplay(w, event, region)
    Widget w;
    XEvent *event;
    Region region; /*unused*/
{
  Window window = XtWindow(w);
  PlotWidget pw = (PlotWidget) w;
  PlotPart *plot = &pw->plot;
  CorePart *core = &pw->core;

  GC gc1 = plotClassRec.plot_class.gc1,
    gc2 = plotClassRec.plot_class.gc2,
```

```
    gc3 = plotClassRec.plot_class.gc3,
    gc  = plotClassRec.plot_class.pointerGC;

  XFontStruct *smallFont = plotClassRec.plot_class.small_font;
  register int startX, startY, endX, endY; /* start,end of plotwin widget */
#ifdef SAVEDATA
  double oldestVisibleTime;
#endif SAVEDATA
  int label1pos = 0, label2pos = 0, label3pos = 0;

  if ((event && (event->type != Expose
                 || (event->type == Expose && event->xexpose.count)))
      || !w || !XtIsRealized(w))
    return; /*Skip all region events but last expose (count == 0)*/

  /* draw the widget */

  XSetBackground(dpy, gc1, plot->background);
  XSetBackground(dpy, gc2, plot->background);
  XSetBackground(dpy, gc3, plot->background);

  XClearWindow(dpy, window);

#ifdef PIXDRAW
  XClearWindow(dpy, plot->plotwin);
#endif PIXDRAW

  /* Establish coordinates for graphing and conversion constants */
  /* X-axis related stuff */
  startX = PAD;
  if (plot->place1 > -1) startX += AXISPAD;
  if (plot->place2 > -1) startX += PAD+AXISPAD;
  endX = core->width - AXISPAD;
  if (plot->place3 > -1) endX -= PAD;
  if (plot->typeX == TIME) {
    if (plot->stripMode) plot->maxX = plot->currentTime;
    plot->sfX = plot->pspeed/speed * RESOLUTION;
    plot->minX = plot->maxX - (endX - startX) / plot->sfX;
    /* fix the end of the graph again to fix the roundoff error */
    endX = startX + plot->sfX*(plot->maxX - plot->minX);
  }
  /* Y-axis oriented stuff */
  {
    int top = plot->titlePad;
    int bot = core->height - plot->labelPad;

    /* add the modify axes button and the refresh button to the top */
    plot->xoff = PAD;
    XtMoveWidget(plot->modButton, plot->xoff, top);
    plot->xoff += plot->modButton->core.width + 8;
    XtMoveWidget(plot->refreshButton,plot->xoff,top);
    plot->xoff += plot->refreshButton->core.width + 8;
    top += (plot->modButton->core.height > plot->refreshButton->core.height) ?
           plot->modButton->core.height : plot->refreshButton->core.height;
    if (XtIsManaged(plot->label3w)) {
      bot -= plot->label3w->core.height;
      label3pos = bot + (plot->label3w->core.height + 0.5)/2;
      XtConfigureWidget(plot->label3w, plot->labelPad + SAMPLEPAD, bot,
            core->width - 2*plot->labelPad - SAMPLEPAD, plot->label3w->core.height,0);
      bot -= plot->labelPad;
    }
    if (XtIsManaged(plot->label2w)) {
      bot -= plot->label2w->core.height;
      label2pos = bot + (plot->label2w->core.height + 0.5)/2;
      XtConfigureWidget(plot->label2w, plot->labelPad + SAMPLEPAD, bot,
                        core->width- 2*plot->labelPad - SAMPLEPAD, plot->label2w->core.height,0);
      bot -= plot->labelPad;
    }
    if (XtIsManaged(plot->label1w)) {
      bot -= plot->label1w->core.height;
      label1pos = bot + (plot->label1w->core.height + 0.5)/2;
      XtConfigureWidget(plot->label1w, plot->labelPad + SAMPLEPAD, bot,
            core->width-2*plot->labelPad - SAMPLEPAD, plot->label1w->core.height,0);
      bot -= plot->labelPad;
    }
    bot -= AXISPAD*2;
    startY = bot-AXISPAD;
    endY   = top+AXISPAD;
  }

  {
    int oldw = plot->plotWidth;
    int oldh = plot->plotHeight;

    plot->plotWidth = endX-startX;
    plot->plotHeight = startY-endY;
    plot->slideDots = (endX-startX)*abs(endY-startY)/10000. + 1;
    plot->nextHash = ceil(plot->minX);
    if (plot->plotWidth < 2 || plot->plotHeight < 2) { /***abort display***/
      return;
    }
    XMoveResizeWindow(dpy, plot->plotwin, startX, endY,
                      plot->plotWidth, plot->plotHeight);
#ifdef SAVEDATA
    oldestVisibleTime = plot->currentTime - plot->decayTime;
    if (plot->decayTime == 0.0)
      oldestVisibleTime = 0.0; /* visible forever */
    if (plot->typeX == TIME) oldestVisibleTime = plot->minX;
#endif SAVEDATA
    if (plot->typeX != TIME) {
      plot->sfX = drawAxis(dpy, window, gc, smallFont,
                           startX, startY+AXISPAD, endX, startY+AXISPAD,
                           plot->minX, plot->maxX, 1);
    }
    if (!plot->plotpix || plot->plotWidth != oldw || plot->plotHeight != oldh){
#ifdef PIXDRAW
      Pixmap newpix = XCreatePixmap(dpy, plot->plotwin, plot->plotWidth,
                                    plot->plotHeight,
                                    XDisplayPlanes(dpy, XDefaultScreen(dpy)));
      XFillRectangle(dpy, newpix, whiteGC, 0, 0, plot->plotWidth, plot->plotHeight);
      if (plot->plotpix) XFreePixmap(dpy, plot->plotpix);
      XSetWindowBackgroundPixmap(dpy, plot->plotwin, newpix);
#else PIXDRAW
      Window newpix = plot->plotwin;
#endif PIXDRAW
      /****||||||?????? copy old pixmap to new */
      XClearWindow(dpy, plot->plotwin);
      plot->plotpix = (Drawable) newpix;
    }
  }
  if (plot->place1 > -1) {
    plot->sf1 = drawAxis(dpy, window, gc1, smallFont, startX - AXISPAD, startY,
```

```
                        startX - AXISPAD, endY, plot->min1, plot->max1, 1);
  XDrawImageString(dpy, window, gc,startX-AXISPAD, startY+AXISPAD, "@1", 2);
  XDrawLine(dpy, window, gc1, plot->labelPad, label1pos,
            plot->labelPad + SAMPLEPAD, label1pos);
  plot->lastPt1 = drawOldData(pw, gc1,smallFont, plot->max1, plot->sf1);
 }
 if (plot->place2 > -1) {
  XSetLineAttributes(dpy, gc2, 0, LineSolid, CapButt, JoinMiter);
  plot->sf2 = drawAxis(dpy,window,gc2,smallFont, startX-2*AXISPAD-PAD,startY,
                       startX-2*AXISPAD-PAD,endY, plot->min2, plot->max2, 1);
  XDrawImageString(dpy, window, gc, startX-2*AXISPAD-PAD, startY+AXISPAD,
                   "@2", 2);
  XSetLineAttributes(dpy, gc2, 0, LineOnOffDash, CapButt, JoinMiter);
  XDrawLine(dpy, window, gc2, plot->labelPad, label2pos,
            plot->labelPad + SAMPLEPAD, label2pos);
  plot->lastPt2 = drawOldData(pw, gc2,smallFont, plot->max2, plot->sf2);
 }
 if (plot->place3 > -1) {
  XSetLineAttributes(dpy, gc3, 0, LineSolid, CapButt, JoinMiter);
  plot->sf3 = drawAxis(dpy, window, gc3, smallFont, endX+AXISPAD, startY,
                       endX+AXISPAD, endY, plot->min3, plot->max3, -1);
  XDrawImageString(dpy, window, gc, endX+AXISPAD-5, startY+AXISPAD,
                   "@3", 2);
  XSetLineAttributes(dpy, gc3, 0, LineOnOffDash, CapButt, JoinMiter);
  XDrawLine(dpy, window, gc3, plot->labelPad, label3pos,
            plot->labelPad + SAMPLEPAD, label3pos);
  plot->lastPt3 = drawOldData(pw, gc3,smallFont, plot->max3, plot->sf3);
 }
 XSetBackground(dpy, gc1, plot->plotback);
 XSetBackground(dpy, gc2, plot->plotback);
 XSetBackground(dpy, gc3, plot->plotback);

/* (superclass->core.class.expose)(w, event, region);*/
} /* END Redisplay() */

/*ARGSUSED*/
static void UpdateCaliper(w, client, event)
    Widget w;
    caddr_t client;/*unused*/
    XEvent *event;
{
 Window window = XtWindow(w);
 PlotWidget pw = (PlotWidget) w;
 PlotPart *plot = &pw->plot;
 GC gc = plotClassRec.plot_class.pointerGC;
 XMotionEvent *mov = (XMotionEvent *) event;
 char buf[132], bufX[40], buf1[40], buf2[40], buf3[40];
 int xPt, yPt;
 Window root_ne, child;
 int rootx, rooty;
 unsigned int mask;

if (!XQueryPointer(dpy, plot->plotwin, &rootret, &child, &rootx, &rooty,
                   &xPt, &yPt, &mask) == False) return; /* wrong screen */
if (mov->subwindow == plot->plotwin) {
 if (plot->sfX)
  sprintf(bufX,"X:%-7.3f", xPt/plot->sfX + plot->minX);
 if (plot->place1 > -1 && plot->sf1)
  sprintf(buf1,", Y1:%-7.3f", yPt/plot->sf1 + plot->max1);
 else buf1[0] = (char)0;
 if (plot->place2 > -1 && plot->sf2)
  sprintf(buf2,",  Y2:%-7.3f", yPt/plot->sf2 + plot->max2);
 else buf2[0] = (char)0;
 if (plot->place3 > -1 && plot->sf3)
  sprintf(buf3,",  Y3:%-7.3f", yPt/plot->sf3 + plot->max3);
 else buf3[0] = (char)0;
 sprintf(buf,"%s%s%s%s", bufX, buf1, buf2, buf3);
 XDrawImageString(dpy, window, gc, plot->xoff, plot->titlePad + 20,
                  buf, strlen(buf));
 } else
  EraseCaliper(w, client, event);
}

/*ARGSUSED*/
static void WriteCaliper(w, client, event)
    Widget w;
    caddr_t client; /*unused*/
    XEvent *event;
{
 Window window = XtWindow(w);
 PlotWidget pw = (PlotWidget) w;
 PlotPart *plot = &pw->plot;
 GC gc = plotClassRec.plot_class.pointerGC;
 XButtonEvent *bev = (XButtonEvent *) event;
 char buf[132], bufX[40], buf1[40], buf2[40], buf3[40];
 int xPt, yPt;
 Window child;

 if (bev->subwindow == plot->plotwin &&
    XTranslateCoordinates(dpy, window, plot->plotwin, bev->x, bev->y,
                          &xPt, &yPt, &child)) {
  XSetBackground(dpy, plotClassRec.plot_class.pointerGC, plot->plotback);
  if (plot->sfX)
   sprintf(bufX,"X:%-.3f", xPt/plot->sfX + plot->minX);
  if (plot->place1 > -1 && plot->sf1)
   sprintf(buf1," Y1:%-.3f", yPt/plot->sf1 + plot->max1);
  else buf1[0] = (char)0;
  if (plot->place2 > -1 && plot->sf2)
   sprintf(buf2," Y2:%-.3f", yPt/plot->sf2 + plot->max2);
  else buf2[0] = (char)0;
  if (plot->place3 > -1 && plot->sf3)
   sprintf(buf3," Y3:%-.3f", yPt/plot->sf3 + plot->max3);
  else buf3[0] = (char)0;
  sprintf(buf,"%s%s%s%s", bufX, buf1, buf2, buf3);
  XDrawLine(dpy, plot->plotpix, gc, xPt-10, yPt, xPt+10, yPt);
  XDrawLine(dpy, plot->plotpix, gc, xPt, yPt-10, xPt, yPt+10);
  xPt += 10;
  if (yPt < 15) yPt += 18;
  else yPt -= 10;
  XDrawImageString(dpy, plot->plotpix, gc, xPt, yPt, buf, strlen(buf));
  XSetBackground(dpy, plotClassRec.plot_class.pointerGC, plot->background);
 }
}

/*ARGSUSED*/
static void EraseCaliper(w, client, event)
    Widget w;
    caddr_t client; /*unused*/
    XEvent *event;
{
 Window window = XtWindow(w);
 PlotWidget pw = (PlotWidget) w;
```

```
  PlotPart *plot = &pw->plot;
  CorePart *core = &pw->core;

  /* Clear text including descenders, but don't bite into axes. */
  XClearArea(dpy, window, 0, 0, (int)core->width, plot->titlePad+20+5, False);
}

/*ARGSUSED*/
static int drawOldData(pw, gc, font, maxY, ofY)
PlotWidget pw;
GC gc;
XFontStruct *font;
double maxY, ofY;
{
  return(-1);
}

#ifdef MOVINGBALL
/*ARGSUSED*/
static void moveBall(w, x1, y1, x2, y2)
Drawable w;
int x1, y1, x2, y2;
{
  GC invert_gc = plotClassRec.plot_class.pointerGC;
/*||| A travesty! This will slow it down miserably!!!!!!!  ||||||*/
  XFillRectangle(dpy, w, invert_gc, x2-1, y2-1, 3, 3);
  XFlush(dpy);
  XFillRectangle(dpy, w, invert_gc, x2-1, y2-1, 3, 3);
}
#endif MOVINGBALL

/*
 * ModCallback() is the action for the modify axes button.
 */
/*ARGSUSED*/
static void ModCallback(widget, closure, call_data)
Widget widget;
caddr_t closure, call_data;
{
  PlotWidget pw = (PlotWidget) closure;
  PlotPart *plot = &pw->plot;
  Widget menu;
  AxisDataRec adX, ad1, ad2, ad3;

  adX.paperSpeed = plot->paperSpeed;
  adX.place = plot->placeX;
  adX.type = plot->typeX;
  adX.title = plot->labelX;
  adX.min = plot->minX;
  adX.max = plot->maxX;
  adX.units = plot->unitX;
  adX.autoScale = plot->autoX;
  adX.scrapMode = plot->scrapMode;
  ad1.place = plot->place1;
  ad1.type = plot->type1;
  ad1.title = plot->label1;
  ad1.min = plot->min1;
  ad1.max = plot->max1;
  ad1.units = plot->unit1;
  ad1.autoScale = plot->auto1;
  ad2.place = plot->place2;
  ad2.type = plot->type2;
  ad2.title = plot->label2;
  ad2.min = plot->min2;
  ad2.max = plot->max2;
  ad2.units = plot->unit2;
  ad2.autoScale = plot->auto2;
  ad3.place = plot->place3;
  ad3.type = plot->type3;
  ad3.title = plot->label3;
  ad3.min = plot->min3;
  ad3.max = plot->max3;
  ad3.units = plot->unit3;
  ad3.autoScale = plot->auto3;

  menu = MakePlotMenu((Widget)pw, XtSetValues, 500, 5,
                      &adX, &ad1, &ad2, &ad3);
#ifdef DEBUG
  printf("Pushed Modify Button ...\n");
#endif
  XtPopup(menu, XtGrabExclusive);
}

/*ARGSUSED*/
static void RefreshCallback(widget, closure, call_data)
Widget widget;
caddr_t closure, call_data;
{
  PlotPart *plot = &((PlotWidget)closure)->plot;
  /* ||| XPixFill plot->plotWin to BlackPixel would give a nice effect */
  XClearWindow(dpy, plot->plotwin);
     /* Clear here, since we won't do it on Redisplay any more */
  plot->lastPtX = plot->lastPt1 = plot->lastPt2 = plot->lastPt3 = -1;
  Redisplay((Widget)closure, (XEvent *)NULL, (Region)NULL);
}

/*ARGSUSED*/
static Boolean SetValues(prev, request, new_widget)
    Widget prev, request, new_widget;
{
  PlotPart *old = & ((PlotWidget)prev)->plot;
  PlotPart *new  = & ((PlotWidget)new_widget)->plot;
  Arg arg[10];
  int i;

  if (old->place3 != new->place3 || old->type3 != new->type3 ||
     old->place2 != new->place2 || old->type2 != new->type2 ||
     old->place1 != new->place1 || old->type1 != new->type1 ||
     old->placeX != new->placeX || old->typeX != new->typeX) {
   }

  if (new->place1 > -1) {
   char buf[1024];
   if (new->typeX == TIME)
    sprintf(buf, "@1:   %s (%s) v.   %s (%g %s)",
            new->label1, new->unit1, new->labelX, (double)new->paperSpeed,
            new->unitX);
   else
    sprintf(buf, "@1:   %s (%s) v.   %s (%s)",
            new->label1, new->unit1, new->labelX, new->unitX);
   i=0;
   XtSetArg(arg[i], XtNlabel, buf);i++;
```

```
  XtSetValues(new->label1w, arg, i);
  XtManageChild(new->label1w);
 } else {
  new->label1 = NULL;
  XtUnmanageChild(new->label1w);
 }

 if (new->place2 > -1) {
  char buf[1024];
  if (new->typeX == TIME)
   sprintf(buf, "@2:   %s (%s) v.   %s (%g %s)",
           new->label2, new->unit2, new->labelX, (double)new->paperSpeed,
           new->unitX);
  else
   sprintf(buf, "@2:   %s (%s) v.   %s (%s)",
           new->label2, new->unit2, new->labelX, new->unitX);
  i=0;
  XtSetArg(arg[i], XtNlabel, buf);i++;
  XtSetValues(new->label2w, arg, i);
  XtManageChild(new->label2w);
 } else {
  new->label2 = NULL;
  XtUnmanageChild(new->label2w);
 }

 if (new->place3 > -1) {
  char buf[1024];
  if (new->typeX == TIME)
   sprintf(buf, "@3:   %s (%s) v.   %s (%g %s)",
           new->label3, new->unit3, new->labelX, (double)new->paperSpeed,
           new->unitX);
  else
   sprintf(buf, "@3:   %s (%s) v.   %s (%s)",
           new->label3, new->unit3, new->labelX, new->unitX);
  i=0;
  XtSetArg(arg[i], XtNlabel, buf);i++;
  XtSetValues(new->label3w, arg, i);
  XtManageChild(new->label3w);
 } else {
  new->label3 = NULL;
  XtUnmanageChild(new->label3w);
 }
 new->lastPtX = new->lastPt1 = new->lastPt2 = new->lastPt3 = -1;
 new->eraseFlash = 0;
 if (new->stripMode != old->stripMode) {
  new->slideData = 1;
  new->cntX = new->cnt1 = new->cnt2 = new->cnt3 = 0;
  new->lastX = new->last1 = new->last2 = new->last3 = 0.;
 }
 return(True); /* always redisplay */
}

/*ARGSUSED*/
static XtGeometryResult GeometryManager(w, request, geometry_return)
    Widget w;
    XtWidgetGeometry *request;
    XtWidgetGeometry *geometry_return;
{
 return(XtGeometryYes); /* always allow subwidgets to resize themselves */
}

/*****************************************************************
 *
 * Public Procedures
 *
 *****************************************************************/

/*ARGSUSED*/
void PlotModifySimulation(w, place, type, value)
    Widget w;
    int place, type;
    double value;
{
/* No longer used. */
#ifdef DEBUG
 printf("Setting p:%d, t:%d, v:%lf\n",place,type,value);
#endif
}

/*
 *
 * Plot New Data event handler
 * Could stand to be cleaned up for execution speed!
 */
/*PUBLIC*/
void PlotData(w, pressures)
    Widget w;
    double *pressures;
{
 register PlotPart *plot = &((PlotWidget)w)->plot;
 double x, y1 = 0.0, y2 = 0.0, y3 = 0.0;
 register int xPt, yPt;
 double slideInc; /* in seconds */
#ifdef SAVEDATA
 double eraseTime;
#endif SAVEDATA
 GC gc1 = plotClassRec.plot_class.gc1,
    gc2 = plotClassRec.plot_class.gc2,
    gc3 = plotClassRec.plot_class.gc3;

 if (!plot->plotpix) /* Not yet realized. */
  return;

 if (!UpdatePeriod(pressures, &plot->timing)
     && ((plot->timing&0x1800) != 0x1800)) {
  if (plot->timing & 0x80 && !(plot->timing&0x800))
   plot->lastPt1 = plot->lastPt2 = plot->lastPt3 = -1;
  return; /* Only plot interesting parts of the cycle */
 }

 if (plot->timing & 0x0800) {/* Plotting average instead of direct value */
  plot->cntX++; plot->cnt1++; plot->cnt2++; plot->cnt3++;
  if (plot->placeX > -1)
   plot->sumX += getval(pressures, plot->placeX, plot->typeX);
  if (plot->place1 > -1)
   plot->sum1 += getval(pressures, plot->place1, plot->type1);
  if (plot->place2 > -1)
   plot->sum2 += getval(pressures, plot->place2, plot->type2);
  if (plot->place3 > -1)
   plot->sum3 += getval(pressures, plot->place3, plot->type3);
  if (plot->timing & 0x1000) { /* Start of beat, so plot it */
   plot->sumX /= plot->cntX; plot->cntX = 0;
```

```
    plot->sum1 /= plot->cnt1; plot->cnt1 = 0;
    plot->sum2 /= plot->cnt2; plot->cnt2 = 0;
    plot->sum3 /= plot->cnt3; plot->cnt3 = 0;
  } else
    return; /* Wait until the start of the next beat. */
}
if (plot->typeX != TIME && (plot->timing & 0x0800))
  { x=plot->sumX; plot->sumX=0.0; }
else
  x=getval(pressures, plot->placeX, plot->typeX);

if (plot->typeX == TIME && plot->currentTime > x)
  plot->nextHash = plot->currentTime;
plot->currentTime = getval(pressures, SYSTEM4, TIME);

if (plot->typeX == TIME) {
  if (plot->lastPtX < 0) {
    /* Redefine the X axis */
    plot->maxX = x;
    plot->sfX = plot->paperSpeed * RESOLUTION;
    plot->minX = plot->maxX - plot->plotWidth / plot->sfX;
    RedisplayAll(w, (XEvent *)NULL, (Region)NULL);
  }
  if (x >= plot->maxX) { /* Off right edge of plot */
    if (plot->stripMode) {
        /* Strip Chart mode: slide image to left by slideDots dots */
        int diff;
        slideInc = plot->slideDots / plot->sfX;
        while ((diff= plot->sfX*(x - plot->minX)
                    - (plot->plotWidth+plot->slideDots)) > 0) {
          /* make sure x will be visible */
          plot->slideDots+= diff;
          slideInc = plot->slideDots / plot->sfX;
        }
        plot->minX += slideInc;
        plot->maxX += slideInc;
        plot->lastPtX -= plot->slideDots;

        XCopyArea(dpy, plot->plotpix, plot->plotpix, gc1, plot->slideDots,
                  0, plot->plotWidth - plot->slideDots, plot->plotHeight,
                  0, 0);
#ifdef PIXDRAW
        XFillRectangle(dpy, plot->plotpix, whiteGC,
                       plot->plotWidth - plot->slideDots,
                       0, plot->slideDots, plot->plotHeight /*full height*/);
#else !PIXDRAW
        XClearArea(dpy,(Window)plot->plotpix,
                   plot->plotWidth-plot->slideDots,
                   0, plot->slideDots, 0 /* full height */ , False);
#endif PIXDRAW
#ifdef NOTDEF
        /*|||??? do we need?  Some implementations write new before copying */
        XFlush(dpy);
#endif NOTDEF
    } else /* Not stripMode */ {
        /* Oscilloscope scan mode: restart at beginning of plot */
        slideInc = plot->maxX - plot->minX;
        plot->minX += slideInc;
        plot->maxX += slideInc;
        plot->lastPt1 = plot->lastPt2 = plot->lastPt3 = -1;
    }
  }
#ifdef SAVEDATA
  eraseTime = plot->currentTime - plot->decayTime;
  if (plot->decayTime == 0.0) eraseTime = 0.0; /* never erase */
#endif SAVEDATA
  xPt = plot->sfX * (x - plot->minX);

  if (plot->typeX == TIME  && !plot->stripMode) {
    /* Erase a band of old data just in front of the current data */
    if (plot->lastPtX > xPt) {
#ifdef PIXDRAW
      XFillRectangle(dpy,plot->plotpix, whiteGC, 0, 0, xPt+6, plot->plotHeight);
#else PIXDRAW
      XClearArea(dpy, (Window) plot->plotpix, 0, 0, xPt + 6, 0, False);
#endif PIXDRAW
      if (plot->lastPtX > 0)
          plot->lastPtX = 0; /* start over from left side of screen */
      else
          plot->lastPtX = xPt; /* but not if we aren't going to be there */
    } else {
      if (xPt != plot->lastPtX) {
#ifdef PIXDRAW
          XFillRectangle(dpy, plot->plotpix, whiteGC, plot->lastPtX+6, 0,
                         plot->slideDots, plot->plotHeight);
#else PIXDRAW
          XClearArea(dpy, (Window) plot->plotpix, plot->lastPtX+6, 0,
                     xPt-plot->lastPtX, 0 /*entire height*/ ,False);
#endif PIXDRAW
      }
    }
  }
  if (plot->lastPtX < 0) plot->lastPtX = xPt;

  if (plot->place1 > -1) {
    if (plot->timing&0x0800)
      { y1 = plot->sum1; plot->sum1 = 0.0; }
    else
      y1 = getval(pressures, plot->place1, plot->type1);
    yPt = plot->sf1 * (y1 - plot->max1);
    if (plot->lastPtX != xPt || plot->lastPt1 != yPt) {
      if (plot->lastPt1 == -1) { plot->lastPt1 = yPt; plot->lastPtX = xPt; }
      XDrawLine(dpy, plot->plotpix, gc1, plot->lastPtX,plot->lastPt1,xPt, yPt);
      plot->lastPt1 = yPt;
#ifdef MOVINGBALL
      moveBall(plot->plotpix, plot->lastPtX, plot->lastPt1, xPt, yPt);
#endif MOVINGBALL
    }
  }
  if (plot->place2 > -1) {
    if (plot->timing&0x0800)
      { y2 = plot->sum2; plot->sum2 = 0.0; }
    else
      y2 = getval(pressures, plot->place2, plot->type2);
    yPt = plot->sf2 * (y2 - plot->max2);
    if (plot->lastPtX != xPt || plot->lastPt2 != yPt) {
      if (plot->lastPt2 == -1) { plot->lastPt2 = yPt; plot->lastPtX = xPt; }
      if ( abs(xPt - plot->lastPtX) > abs(yPt - plot->lastPt2))
          plot->phase2 = (plot->phase2 + abs(xPt - plot->lastPtX)) % 4;
      else
```

```
            plot->phase2 = (plot->phase2 + abs(yPt - plot->lastPt2)) % 4;
          XSetDashes(dpy, gc2, plot->phase2, Dash2, strlen(Dash2));
          XDrawLine(dpy,plot->plotpix, gc2, plot->lastPtX,plot->lastPt2, xPt, yPt);
          plot->lastPt2 = yPt;
#ifdef MOVINGBALL
          moveBall(plot->plotpix, plot->lastPtX, plot->lastPt2, xPt, yPt);
#endif MOVINGBALL
        }
      }
      if (plot->place3 > -1) {
        if (plot->timing&0x0800)
          { y3 = plot->sum3; plot->sum3 = 0.0; }
        else
          y3 = getval(pressure, plot->place3, plot->type3);
        yPt = plot->sf3 * (y3 - plot->min3);
        if (plot->lastPtX != xPt || plot->lastPt3 != yPt) {
          if (plot->lastPt3 == -1) { plot->lastPt3 = yPt; plot->lastPtX = xPt; }
          if ( abs(xPt - plot->lastPtX) > abs(yPt - plot->lastPt3))
              plot->phase3 = (plot->phase3 + abs(xPt - plot->lastPtX)) % 8;
          else
              plot->phase3 = (plot->phase3 + abs(yPt - plot->lastPt3)) % 8;
          XSetDashes(dpy, gc3, plot->phase3, Dash3, strlen(Dash3));
          XDrawLine(dpy,plot->plotpix, gc3, plot->lastPtX,plot->lastPt3, xPt, yPt);
          plot->lastPt3 = yPt;
#ifdef MOVINGBALL
          moveBall(plot->plotpix, plot->lastPtX, plot->lastPt3, xPt, yPt);
#endif MOVINGBALL
        }
      }
      plot->lastPtX = xPt;

      if (plot->typeX == TIME && plot->nextHash + 0.1 <= x) {
        xPt = plot->offX*(plot->nextHash - plot->minX);
        yPt = plot->plotHeight;
        XDrawLine(dpy, plot->plotpix, plotClassRec.plot_class.pointerGC,
                  xPt, yPt, xPt, yPt-3);
        plot->nextHash += 1;
      }

      if (plot->autoX) {
        switch (plot->outX) {
        case 0:
          if (x > plot->maxX) plot->outX = 1;
          else if (x < plot->minX) plot->outX = -1;
          break;
        case -1:
          if (plot->lastX < y1) {
              plot->minX = plot->lastX - (plot->maxX-plot->lastX) * .1;
              plot->outX = 0;
              Redisplay(w, (XEvent *)NULL, (Region)NULL);
          }
          break;
        case 1:
          if (plot->lastX > y1) {
              plot->maxX = plot->lastX + (plot->lastX-plot->minX) * .1;
              plot->outX = 0;
              Redisplay(w, (XEvent *)NULL, (Region)NULL);
          }
          break;
        }
        plot->lastX = x;
      }
      /*
       * Check for need to automatically rescale
       */
      if (plot->auto1) {
        switch (plot->out1) {
        case 0:
          if (y1 > plot->max1) plot->out1 = 1;
          else if (y1 < plot->min1) plot->out1 = -1;
          break;
        case -1:
          if (plot->last1 < y1) {
              plot->min1 = plot->last1 - (plot->max1-plot->last1) * .1;
              plot->out1 = 0;
              Redisplay(w, (XEvent *)NULL, (Region)NULL);
          }
          break;
        case 1:
          if (plot->last1 > y1) {
              plot->max1 = plot->last1 + (plot->last1-plot->min1) * .1;
              plot->out1 = 0;
              Redisplay(w, (XEvent *)NULL, (Region)NULL);
          }
          break;
        }
        plot->last1 = y1;
      }

      if (plot->auto2) {
        switch (plot->out2) {
        case 0:
          if (y2 > plot->max2) plot->out2 = 1;
          else if (y2 < plot->min2) plot->out2 = -1;
          break;
        case -1:
          if (plot->last2 < y2) {
              plot->min2 = plot->last2 - (plot->max2-plot->last2) * .1;
              plot->out2 = 0;
              Redisplay(w, (XEvent *)NULL, (Region)NULL);
          }
          break;
        case 1:
          if (plot->last2 > y2) {
              plot->max2 = plot->last2 + (plot->last2-plot->min2) * .1;
              plot->out2 = 0;
              Redisplay(w, (XEvent *)NULL, (Region)NULL);
          }
          break;
        }
        plot->last2 = y2;
      }

      if (plot->auto3) {
        switch (plot->out3) {
        case 0:
          if (y3 > plot->max3) plot->out3 = 1;
          else if (y3 < plot->min3) plot->out3 = -1;
          break;
        case -1:
          if (plot->last3 < y3) {
```

```
        plot->min3 = plot->last3 - (plot->max3-plot->last3) * .1;
        plot->out3 = 0;
        Redisplay(w, (XEvent *)NULL, (Region)NULL);
    }
    break;
  case 1:
    if (plot->last3 > y3) {
        plot->max3 = plot->last3 + (plot->last3-plot->min3) * .1;
        plot->out3 = 0;
        Redisplay(w, (XEvent *)NULL, (Region)NULL);
    }
    break;
  }
  plot->last3 = y3;
 }
#ifdef PIXDRAW
  XClearWindow(dpy, plot->plotwin);
#endif PIXDRAW
} /* END of PlotData() */
```

drawAxis.c

```
#ifndef lint
static char *sccs="$Header: /usr/hal/src/plot/RCS/drawAxis.c,v 2.1 90/08/18 18:37:59 tdavis Exp $";
#endif
/*Copyright 1988 by Timothy L. Davis
 *
 *Draws Axis in either horizontal or vertical direction with hash marks
 *as requested.  For use by the Plot widget.
 *
 *$Source: /usr/hal/src/plot/RCS/drawAxis.c,v $
 *$Log:  drawAxis.c,v $
 * Revision 2.1  90/08/18  18:37:59  tdavis
 * Small bugfixes?
 *
 * Revision 2.0  90/08/06  18:00:05  tdavis
 * First X11R3 version.
 *
 * Revision 1.6  89/02/22  12:27:28  tdavis
 * X11R3 version (no saves of old data).
 *
 * Revision 1.5  88/10/10  01:22:21  tdavis
 * Fixed positioning of numbers.
 *
 * Revision 1.4  88/10/04  08:21:59  tdavis
 * *** empty log message ***
 *
 * Revision 1.3  88/08/16  16:03:11  tdavis
 * *** empty log message ***
 *
 * Revision 1.2  88/08/03  18:22:20  tdavis
 * First successful compile of X11 version.
 *
 * Revision 1.1  88/08/03  14:54:55  tdavis
 * Initial revision
 *
 */
#include <math.h>
#include <stdio.h>
#include <X11/Xlib.h>

#define MIN_SPACE 4

static double myfabs(x)
    double x;
{
 return((((x) > -0.001 && (x) < 0.001) ? 1.0 : fabs(x)));
}

/*PUBLIC*/ double
drawAxis(dpy, window, gc, smallFont, startX, startY, endX, endY, min, max, tic)
Display *dpy;
Window window;
GC gc;
XFontStruct *smallFont;
int startX, startY, endX, endY;
double min, max;
int tic;
{
 double ticint, xIX, xIY;
 double order, temp;
 int x, y; /* screen coordinates */
 double next;
 char s[80];
 int xNumOffset, yNumOffset;
 int nextnumloc, lastnumloc;
 int font_width = smallFont ? XTextWidth(smallFont, "0", 1) : 5;
 int font_height = 10;

 if (startX==endX) ticint = fabs((max - min)*MIN_SPACE*5./(endY-startY));
 else if (startY==endY) ticint = (max - min)*MIN_SPACE*5./(endX-startX);
 else { fprintf(stderr, "in drawAxis:   Axis must not be diagonal\n"); }

 xIX = (endX-startX)/(max - min);
 xIY = (endY-startY)/(max - min);

 /* determine tic interval based on a round number (10, 25, 50...) */
 order = pow(10.0, floor(log10(ticint)));
 temp = ticint/order;
 if (temp <= 1.0001) temp = 1.;
 else if (temp <= 2.5001) temp = 2.5;
 else if (temp <= 5.0001) temp = 5.0;
 else temp = 10.0;
 ticint = temp*order;
 if (startX==endX) {
  xNumOffset = (tic>0 ? -tic*5*font_width : 0) - tic*5;
  yNumOffset = font_height/2;
 } else { /* startY must == endY */
  xNumOffset = -2*font_width;
  /*||| trying to get x axis right.  */
  yNumOffset = (tic<0 ? tic*font_height : font_height) + tic*5 ;
  nextnumloc = startX + 2*font_width;
  lastnumloc = endX - (4+3)*(font_width);
 }

 sprintf(s, "%4.2f      ",min);
 XDrawImageString(dpy,window,gc,startX+xNumOffset, startY+yNumOffset, s, 5);
 sprintf(s, "%4.2f      ",max);
 XDrawImageString(dpy,window,gc,endX+xNumOffset, endY+yNumOffset, s, 5);
 XDrawLine(dpy,window,gc, startX, startY, endX, endY);
 XDrawLine(dpy,window,gc, startX, startY,(int)(startX-xIY/myfabs(xIY)*tic*8),
```

```
            (int)(stattY-sfX/myfabs(sfX)*tic*8.));
  XDrawLine(dpy,window,gc, endX, endY, (int)(endX-sfY/myfabs(sfY)*tic*8.),
            (int)(endY-sfX/myfabs(sfX)*tic*8.));


  for (next=ticint*ceil(min/ticint); next < max; next += ticint) {
    x = (next-min)*sfX+startX;
    y = (next-min)*sfY+startY;
    XDrawLine(dpy,window,gc, x, y,(int)(x-sfY/myfabs(sfY)*tic*4.),
              (int)(y-sfX/myfabs(sfX)*tic*4.));
    x += xNumOffset;
    y += yNumOffset;
    if (startX==endX || (x >= nextxmaloc && x <= lastxmaloc)) {
          sprintf(s, "%f       ", next);
          XDrawImageString(dpy,window,gc, x, y, s, 4);
          nextxmaloc = x+5*font_width;
      }
  }
  ticint *= 0.2;
  for (next=ticint*ceil(min/ticint); next < max; next += ticint) {
    x = (next-min)*sfX+startX;
    y = (next-min)*sfY+startY;
    XDrawLine(dpy,window,gc, x, y,(int)(x-sfY/myfabs(sfY)*tic*2.),
              (int)(y-sfX/myfabs(sfX)*tic*2.));
  }
  return(sfX ? sfX : sfY);
}
```

C.6 Plot form window

These files define a form window which can be used for the user to fill out information for creating a plot. This is not a widget type definition; rather, Athena widgets are composed into a more specific user interface object.

PlotForm.h

```
/**********************************************************************
 *
 * PlotForm.h: The following public procedures are defined for
 * creating forms for axis information choice by the user.
 *
 **********************************************************************/

typedef struct {
  int place, type;
  char *title;
  double min;
  double max;
  double prepadpoint;
  char *units;
  int autoScale;
  int stripMode;
} *AxisData, AxisDataRec;

/* Create a menu to choose axes for a new plot, user supplied create routine */
extern Widget MakePlotMenu(); /* parent, widgetClass, x, y, circuitDiagram
                                 AxisData X, Y, Z, 3 pointers may be NULL */

extern void replaceText();
```

PlotForm.c

```
/*
 * Plot Form composite widget for X11R4 Cardiovascular Simulator
 * Copyright 1988 by Timothy L. Davis
 *
 * PlotForm is the form which is used to choose variables to be plotted and
 * their axes/labels and other plotting parameters.
 * There are four variable "frames" in which information can be entered.
 * The bottom of the menu provides an overall parameter, the length of time
 * before a plot begins to be erased, and completion buttons.
 * Note: This is NOT a widget definition, but an instance of widget usage.
 *
 * #defines:  SAVEDATA to add a "persist" label and value to the menu
 *            XAW_BC for (probable) backward compatibility to X11R3
 *            DO_GRABS if you wish to grab the server during menu popup.
 *
 * $Source: /usr/lut/src/form/BCS/PlotForm.c,v $
 * $Log:            PlotForm.c,v $
 * Revision 3.1  90/08/28  17:33:50  tldavis
 * small fix.
 *
 * Revision 3.0  90/08/28  14:38:43  tldavis
 * Added buttons for setting averaging and partial cardiac cycle plotting.
 *
 * Revision 2.4  90/08/18  18:33:02  tldavis
 * General cleanup.  Also added formatting, removed specific formatting
 * to use resource file instead.  Fixed a few fmUnders bugs.
 *
 * Revision 2.2  90/08/10  17:31:46  tldavis
 * Fixing problems with interacting with the text in text widgets.
 * 1) Text widgets get "enablePlace" to allow for reading of text values
 * 2) Use XtSetValues instead of XawReplaceText to put text in widget.
 *
 * Revision 2.1  90/08/10  10:53:31  dat
 * upgrade to R4; mostly changes to the ascii text widgets
 * new code ifdef'd under !XAW_BC
 *
 * Revision 2.0  90/08/06  17:53:25  tldavis
 * First X11R3 version.
 *
 * Revision 1.16  90/08/06  17:07:50  tldavis
 * Fixed a problem with color.
 * Revision 1.14  90/05/31  16:54:09  tldavis
 * Pre-ACSE checkin.
 * Revision 1.13  90/10/12  22:13:16  tldavis
 * Fixed for Motif window manager plus indep. var. plottability.
 * Revision 1.11  90/06/28  11:48:46  tldavis:removed persist, added stripMode.
 * Revision 1.10  90/06/13  21:24:47  tldavis : R3 fonts fixed.
 * Revision 1.9  89/02/22  12:23:22  tldavis : X11R3 version.
 * Revision 1.8  88/10/10  21:36:48  lut
 * Fixed several bugs, notably resize subwidgets.
 */
#include <stdio.h>
#include <math.h>
#ifdef XAW_BC
#include <X11/IntrinsicP.h>
#include <X11/Xaw/TextP.h> /* necessary to get XtTextBlock definition! */
#endif XAW_BC
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
```

```
#include <X11/Shell.h>
#include <X11/Xaw/Form.h>
#include <X11/Xaw/Box.h>
#include <X11/Xaw/Label.h>
#include <X11/Xaw/Command.h>
#include <X11/Xaw/Toggle.h>
#include <X11/Xaw/Paned.h>
#include <X11/Xaw/AsciiText.h>
#include <X11/cursorfont.h>
#include "../plot/Plot.h"
#include "../sim/CVDefs.h"
#include "../data/Parameter.h"
#include "PlotForm.h"
#include "../event/Event.h" /* for extern int Stopped */

extern Display *dpy;
extern Window  root;
extern int screen;
extern int patientActive; /* from Interface.c */
extern Widget BUTTON_BOX; /* from ../main/Interface.c */

/*****************************************************************
 *
 * PRIVATE DATA
 *
 *****************************************************************/

typedef struct {
  Widget form;
  Widget axisLabel, varLabel, nameLabel, minLabel, maxLabel, speedLabel,
         scaleLabel, titleLabel, unitLabel, modeLabel;
  Widget minText, maxText, titleText, scaleButton, unitText, circuitButton,
         speedText, hideButton, modeButton;
  char *minString, *maxString, *speedString, *titleString, *unitString;
  Boolean scaleValue, modeValue;
  int place, type;
} *PFData, PFDataRec;

static Boolean FormActive = False;

/* The first 10 elements represent number key presses; the intent of this
 * translation table is to allow only numeric entry on only 1 line of text.
 */
static String numericTextEventTable =
  "<Key>0x030:      kill-to-end-of-line() beginning-of-file() end-of-line() \
            insert-char() \n\
    <Key>0x031:     kill-to-end-of-line() beginning-of-file() end-of-line() \
            insert-char() \n\
    <Key>0x032:     kill-to-end-of-line() beginning-of-file() end-of-line() \
            insert-char() \n\
    <Key>0x033:     kill-to-end-of-line() beginning-of-file() end-of-line() \
            insert-char() \n\
    <Key>0x034:     kill-to-end-of-line() beginning-of-file() end-of-line() \
            insert-char() \n\
    <Key>0x035:     kill-to-end-of-line() beginning-of-file() end-of-line() \
            insert-char() \n\
    <Key>0x036:     kill-to-end-of-line() beginning-of-file() end-of-line() \
            insert-char() \n\
    <Key>0x037:     kill-to-end-of-line() beginning-of-file() end-of-line() \
            insert-char() \n\
    <Key>0x038:     kill-to-end-of-line() beginning-of-file() end-of-line() \
            insert-char() \n\
    <Key>0x039:     kill-to-end-of-line() beginning-of-file() end-of-line() \
            insert-char() \n\
    <Key>.:   kill-to-end-of-line() beginning-of-file() end-of-line() \
            insert-char() \n\
    <Key>-:   kill-to-end-of-line() beginning-of-file() end-of-line() \
            insert-char() \n\
    Ctrl<Key>F:     forward-character() \n\
    <Key>0xff53:    forward-character() \n\
    Ctrl<Key>B:     backward-character() \n\
    <Key>0xff51:    backward-character() \n\
    Ctrl<Key>A:     beginning-of-line() \n\
    Ctrl<Key>E:     end-of-line() \n\
    Ctrl<Key>P:     previous-line() \n\
    <Key>0xff52:    previous-line() \n\
    Meta<Key>\\<:   beginning-of-file() \n\
    Ctrl<Key>Z:     scroll-one-line-up() \n\
    Ctrl<Key>D:     delete-next-character() \n\
    Ctrl<Key>H:     delete-previous-character() \n\
    <Key>0xff7f:    delete-previous-character() \n\
    <Key>0xffff:    delete-previous-character() \n\
    <Key>0xff08:    delete-previous-character() \n\
    Meta<Key>D:     delete-next-word() \n\
    Meta<Key>H:     delete-previous-word() \n\
    Shift Meta<Key>D: kill-word() \n\
    Shift Meta<Key>H: backward-kill-word() \n\
    Ctrl<Key>W:     kill-selection() \n\
    Ctrl<Key>K:     kill-to-end-of-line() \n\
    Meta<Key>K:     kill-to-end-of-paragraph()";

static XtTranslations parsedNumeric;

/*****************************************************************
 *
 * Private Procedures
 *
 *****************************************************************/
/*FORWARD*/ static void UpdateAxisForm();

/* callback routine for toggling scale button label*/
/*ARGSUSED*/
static void ScaleButtonCallback(w, closure, call)
Widget w;
caddr_t closure, call;
{
  PFData data = (PFData) closure;
  Arg args[2];
  data->scaleValue = !data->scaleValue;
  XtSetArg(args[0], XtNlabel, data->scaleValue ? "Yes" : "No");
  XtSetValues(data->scaleButton, args, 1);
}

/* callback routine for toggling mode button label*/
/*ARGSUSED*/
static void ModeButtonCallback(w, closure, call)
Widget w;
caddr_t closure, call;
{
  PFData data = (PFData) closure;
  Arg args[2];
  data->modeValue = !data->modeValue;
```

```
  XtSetArg(args[0], XtNlabel, data->modeValue ? "StripChart" : "Scope");
  XtSetValues(data->modeButton, args, 1);
}

/* change to NO PARAMETER for this axis */
/*ARGSUSED*/
static void hideButtonCallback(w, closure, call)
Widget w;
caddr_t closure, call;
{
  PFData data = (PFData) closure;
  data->place = -1;
  data->type = -1;
  UpdateAxisForm(data, (char *)NULL, (AxisData)NULL);
}

/* this replaces the text in a Text widget kludgily */
/*PUBLIC*/ void replaceText(widget, text)
Widget widget;
char *text;
{
#ifdef XAW_BC
  XtTextBlock block;
  XtTextPosition top;

  XtTextSetLastPos(widget,top=XtTextTopPosition(widget));

  block.firstPos = 0;
  block.length = strlen(text);
  block.ptr = text;
  block.format = FMTEDIT;

  if (block.length) XtTextReplace(widget, top, top, &block);
  XtTextSetInsertionPoint(widget, top);
#else /* ! XAW_BC */
  Arg args[5];
  int i=0;

  char *tmp = XtMalloc(strlen(text)>40 ? strlen(text)+10 : 40);
  char *oldstr;
  strcpy(tmp, text);

  XtSetArg(args[i], XtNstring, &oldstr); i++;
  XtGetValues(widget, args, i);
  if (oldstr) XtFree(oldstr);
  i=0;
  XtSetArg(args[i], XtNstring, tmp); i++;
  XtSetValues(widget, args, i);
  XawTextSetInsertionPoint(widget, XawTextTopPosition(widget));
#endif /* XAW_BC */
}

/* runs the circuit diagram & fills in data slots; runs on button press */
/*ARGSUSED*/
static void ChannelButtonCallback(widget, closure, call)
    Widget widget;
    caddr_t closure, call;
{
  PFData data = (PFData)closure;
  int place, type;
  Arg args[1];
  VarData *vd;

  xmodify_parameters(patientActive ? 1 : 2, &place, &type);
          /* 1: p,v,f only, 2: p,v,f + indep. parameters */
  if (place < 0 || type < 0) return; /* No update: no choice made */
  data->place = place; data->type = type;
  vd = get_vardata(data->place, data->type);

  strcpy(data->titleString, vd->name);
  replaceText(data->titleText, data->titleString);
  (void) sprintf(data->minString,"%lg", vd->gmin);
  replaceText(data->minText, data->minString);
  (void) sprintf(data->maxString,"%lg", vd->gmax);
  replaceText(data->maxText, data->maxString);
  (void) sprintf(data->speedString,"%lg", 5.0);
  replaceText(data->speedText, data->speedString);

  data->unitsString = vd->units;
  XtSetArg(args[0], XtNlabel, vd->units);
  XtSetValues(data->unitText, args, 1);
  UpdateAxisForm(data, (char *)NULL, (AxisData)NULL);
}

/*ARGSUSED*/
static void DoFocusKeyboard(w, client, event)
    Widget w;
    caddr_t client;
    XEvent *event;
{
 if (XtWindow(w)/*||| && w->core.visible */) {
   XWindowAttributes att;
   XGetWindowAttributes(dpy, XtWindow(w), &att);
   if (att.map_state == IsViewable)
     XSetInputFocus(dpy, XtWindow(w), RevertToParent, CurrentTime);
 }
}

/*
 * routines for single-axis form creation, display, and data cleanup
 * These are usually called by the menu manager later this file.
 */

static PFData CreateAxisForm(parent, passed_args, passed_i)
Widget parent;
Arg *passed_args;
int passed_i;
{
 Arg args[20];
 int i, j;
 PFData data = (PFData) XtMalloc(sizeof(PFDataRec));

 data->form = XtCreateManagedWidget("Axis", formWidgetClass, parent,
                                    passed_args, passed_i);

 parseNumeric = XtParseTranslationTable(numericTextEventTable);
 /*
  * Create a bunch of widget instances to go in the form, and
  * Organize these labels & widgets into a pretty form layout.
  */

 i=0;
```

```
XtSetArg(args[i], XtNtop, XtChainTop);i++;
XtSetArg(args[i], XtNbottom, XtChainTop);i++;
XtSetArg(args[i], XtNleft, XtChainLeft);i++;
XtSetArg(args[i], XtNright, XtChainLeft);i++;
XtSetArg(args[i], XtNvertDistance, 8);i++;
XtSetArg(args[i], XtNresizable, True);i++;
j = i;
/* Row 1 */
XtSetArg(args[i], XtNborderWidth, 0);i++;
XtSetArg(args[i], XtNwidth, 75);i++;
data->axisLabel = XtCreateManagedWidget("AxisLabel", labelWidgetClass,
                                        data->form, args, i);
i=j;
XtSetArg(args[i], XtNfromHoriz, data->axisLabel);i++;
XtSetArg(args[i], XtNborderWidth, 3);i++;
data->circuitButton = XtCreateManagedWidget("Choose Variable",
                        commandWidgetClass, data->form, args, i);
XtAddCallback(data->circuitButton, XtNcallback, CircuitButtonCallback,
              (caddr_t)data);
i=j;
XtSetArg(args[i], XtNfromHoriz, data->circuitButton);i++;
XtSetArg(args[i], XtNhorizDistance, 10);
XtSetArg(args[i], XtNborderWidth, 3);i++;
data->hideButton = XtCreateManagedWidget("Delete", commandWidgetClass,
                                         data->form, args, i);
XtAddCallback(data->hideButton, XtNcallback, hideButtonCallback,
              (caddr_t)data);
/* Row 2 */
i=j;
XtSetArg(args[i], XtNfromVert, data->circuitButton);i++;
XtSetArg(args[i], XtNborderWidth, 0);i++;
data->varLabel = XtCreateManagedWidget("Variable", labelWidgetClass,
                                       data->form, args, i);
i=j;
XtSetArg(args[i], XtNfromVert, data->circuitButton);i++;
XtSetArg(args[i], XtNfromHoriz, data->varLabel);i++;
XtSetArg(args[i], XtNwidth, 200);i++;
XtSetArg(args[i], XtNborderWidth, 0);i++;
data->nameLabel = XtCreateManagedWidget("nameLabel", labelWidgetClass,
                                        data->form, args, i);
/* Row 3 */
i=j;
XtSetArg(args[i], XtNborderWidth, 0);i++;
XtSetArg(args[i], XtNfromVert, data->nameLabel);i++;
data->titleLabel = XtCreateManagedWidget("Title:", labelWidgetClass,
                                         data->form, args, i);
i=j;
XtSetArg(args[i], XtNfromVert, data->nameLabel);i++;
XtSetArg(args[i], XtNfromHoriz, data->titleLabel);i++;
data->titleString = (char *) XtMalloc(40*sizeof(char));
*data->titleString = 0;
XtSetArg(args[i], XtNwidth, 200);i++;
#ifdef XAW_BC
 XtSetArg(args[i], XtNstring, data->titleString);i++;
 XtSetArg(args[i], XtNeditType, XawtextEdit);i++;
 XtSetArg(args[i], XtNscrollOptions, editable);i++;
#else /* ! XAW_BC */
 XtSetArg(args[i], XtNeditType, XawtextEdit);i++;
#endif /* XAW_BC */
 XtSetArg(args[i], XtNlength, 40);i++;
#ifdef XAW_BC
 data->titleText = XtCreateManagedWidget("titleText", asciiStringWidgetClass,
                                         data->form, args, i);
#else /* ! XAW_BC */
 data->titleText = XtCreateManagedWidget("titleText", asciiTextWidgetClass,
                                         data->form, args, i);
#endif /* XAW_BC */
 XtAddEventHandler(data->titleText, EnterWindowMask, False, DoFocusKeyboard, (caddr_t)0);
 /* Row 4 */
 i=j;
 XtSetArg(args[i], XtNfromVert, data->titleText);i++;
 XtSetArg(args[i], XtNborderWidth, 0);i++;
 data->minLabel = XtCreateManagedWidget("Min:", labelWidgetClass,
                                        data->form, args, i);
 i=j;
 XtSetArg(args[i], XtNfromVert, data->titleText);i++;
 XtSetArg(args[i], XtNfromHoriz, data->minLabel);i++;
 data->minString = (char *) XtMalloc(40*sizeof(char));
 *data->minString = 0;
 XtSetArg(args[i], XtNwidth, 75);i++;
#ifdef XAW_BC
 XtSetArg(args[i], XtNstring, data->minString);i++;
 XtSetArg(args[i], XtNeditType, XawtextEdit);i++;
 XtSetArg(args[i], XtNscrollOptions, editable);i++;
#else /* ! XAW_BC */
 XtSetArg(args[i], XtNeditType, XawtextEdit);i++;
#endif /* XAW_BC */
 XtSetArg(args[i], XtNlength, 40);i++;
 XtSetArg(args[i], XtNtranslations, parsedNumeric);i++;
#ifdef XAW_BC
 data->minText = XtCreateManagedWidget("minText", asciiStringWidgetClass,
                                       data->form, args, i);
#else /* ! XAW_BC */
 data->minText = XtCreateManagedWidget("minText", asciiTextWidgetClass,
                                       data->form, args, i);
#endif /* XAW_BC */
 XtAddEventHandler(data->minText, EnterWindowMask, False, DoFocusKeyboard, (caddr_t)0);
 i=j;
 XtSetArg(args[i], XtNfromVert, data->titleText);i++;
 XtSetArg(args[i], XtNfromHoriz, data->minText);i++;
 XtSetArg(args[i], XtNborderWidth, 0);i++;
 data->maxLabel = XtCreateManagedWidget("Max:", labelWidgetClass,
                                        data->form, args, i);
 i=j;
 XtSetArg(args[i], XtNfromVert, data->titleText);i++;
 XtSetArg(args[i], XtNfromHoriz, data->maxLabel);i++;
 data->maxString = (char *) XtMalloc(40*sizeof(char));
 *data->maxString = 0;
 XtSetArg(args[i], XtNwidth, 75);i++;
#ifdef XAW_BC
 XtSetArg(args[i], XtNstring, data->maxString);i++;
 XtSetArg(args[i], XtNeditType, XawtextEdit);i++;
 XtSetArg(args[i], XtNscrollOptions, editable);i++;
#else /* ! XAW_BC */
 XtSetArg(args[i], XtNeditType, XawtextEdit);i++;
#endif /* XAW_BC */
 XtSetArg(args[i], XtNlength, 40);i++;
 XtSetArg(args[i], XtNtranslations, parsedNumeric);i++;
#ifdef XAW_BC
 data->maxText = XtCreateManagedWidget("maxText", asciiStringWidgetClass,
                                       data->form, args, i);
#else /* ! XAW_BC */
```

```
    data->maxText = XtCreateManagedWidget("maxText", asciiTextWidgetClass,
                                          data->form, args, i);
#endif /* XAW_BC */
    XtAddEventHandler(data->maxText, EnterWindowMask, False, DoFocusKeyboard, (caddr_t)0);

    /* Row 4 for TIME */
    i=j;
    XtSetArg(args[i], XtNfromVert, data->titleText);i++;
    XtSetArg(args[i], XtNborderWidth, 0);i++;
    data->speedLabel = XtCreateManagedWidget("Paper Speed", labelWidgetClass,
                                          data->form, args, i);
    i=j;
    XtSetArg(args[i], XtNfromVert, data->titleText);i++;
    XtSetArg(args[i], XtNfromHoriz, data->speedLabel);i++;
    data->speedString = (char *) XtMalloc(40*sizeof(char));
    *data->speedString= 0;
    XtSetArg(args[i], XtNwidth, 75);i++;
#ifdef XAW_BC
    XtSetArg(args[i], XtNstring, data->speedString);i++;
    XtSetArg(args[i], XtNeditType, XttextEdit);i++;
    XtSetArg(args[i], XtNtextOptions, editable);i++;
#else /* ! XAW_BC */
    XtSetArg(args[i], XtNeditType, XawtextEdit);i++;
#endif /* XAW_BC */
    XtSetArg(args[i], XtNlength, 40);i++;
    XtSetArg(args[i], XtNtranslations, parsedNumeric);i++;
#ifdef XAW_BC
    data->speedText = XtCreateManagedWidget("speedText", asciiStringWidgetClass,
                                          data->form, args, i);
#else /* ! XAW_BC */
    data->speedText = XtCreateManagedWidget("speedText", asciiTextWidgetClass,
                                          data->form, args, i);
#endif /* XAW_BC */
    XtAddEventHandler(data->speedText, EnterWindowMask, False, DoFocusKeyboard, (caddr_t)0);
    /* Row 5 */
    i=j;
    XtSetArg(args[i], XtNfromVert, data->maxText);i++;
    XtSetArg(args[i], XtNborderWidth, 0);i++;
    data->unitLabel = XtCreateManagedWidget("Units:", labelWidgetClass,
                                          data->form, args, i);
    i=j;
    XtSetArg(args[i], XtNfromVert, data->maxText);i++;
    XtSetArg(args[i], XtNfromHoriz, data->unitLabel);i++;
    XtSetArg(args[i], XtNlength, 40);i++;
    XtSetArg(args[i], XtNwidth, 75);i++;
    data->unitText = XtCreateManagedWidget("unitText", labelWidgetClass,
                                          data->form, args, i);
    i=j;
    XtSetArg(args[i], XtNfromVert, data->maxText);i++;
    XtSetArg(args[i], XtNfromHoriz, data->unitText);i++;
    XtSetArg(args[i], XtNborderWidth, 0);i++;
    data->scaleLabel = XtCreateManagedWidget("Auto Rescale", labelWidgetClass,
                                          data->form, args, i);
    i=j;
    XtSetArg(args[i], XtNfromVert, data->maxText);i++;
    XtSetArg(args[i], XtNfromHoriz, data->unitText);i++;
    XtSetArg(args[i], XtNborderWidth, 0);i++;
    data->modeLabel = XtCreateManagedWidget("Disp.    Mode:", labelWidgetClass,
                                          data->form, args, i);
    i=j;
    XtSetArg(args[i], XtNfromVert, data->maxText);i++;
    XtSetArg(args[i], XtNfromHoriz, data->scaleLabel);i++;
    data->scaleValue = 0;
    XtSetArg(args[i], XtNlabel, data->scaleValue ? "Yes" : "No" );i++;
    data->scaleButton = XtCreateManagedWidget("scaleButton", commandWidgetClass,
                                          data->form, args, i);
    XtAddCallback(data->scaleButton, XtNcallback, ScaleButtonCallback,
                  (caddr_t)data);
    i=j;
    XtSetArg(args[i], XtNfromVert, data->maxText);i++;
    XtSetArg(args[i], XtNfromHoriz, data->modeLabel);i++;
    data->modeValue = 0;
    XtSetArg(args[i], XtNlabel, data->modeValue ? "Strip Chart" : "Scope" );i++;
    XtSetArg(args[i], XtNwidth, 75);i++;
    data->modeButton = XtCreateManagedWidget("modeButton", commandWidgetClass,
                                          data->form, args, i);
    XtAddCallback(data->modeButton, XtNcallback, ModeButtonCallback,
                  (caddr_t)data);
    return(data);
}

/*
 * Update a previously-created form, optionally updating from axisName
 * or axisData.  if axisName or AxisData are NULL, the previous values
 * for this widget are used.
 */
static void UpdateAxisForm(data , axisName, axisData)
PFData data;
char *axisName;
AxisData axisData;
{
    VmData *vd;
    Arg args[1];

    if (axisName) {
      XtSetArg(args[0], XtNlabel, axisName);
      XtSetValues(data->axisLabel, args, 1);
    }
    if (axisData) {
      data->place = axisData->place;
      data->type = axisData->type;
      replaceText(data->titleText, axisData->title);
      (void) sprintf(data->minString,"%lg", axisData->min);
      replaceText(data->minText, data->minString);
      (void) sprintf(data->maxString,"%lg", axisData->max);
      replaceText(data->maxText, data->maxString);
      (void) sprintf(data->speedString,"%lg", axisData->paperSpeed);
      replaceText(data->speedText, data->speedString);
      XtSetArg(args[0], XtNlabel, axisData->units);
      XtSetValues(data->unitText, args, 1);
      if (data->scaleValue != axisData->autoScale)
        ScaleButtonCallback(data->form, (caddr_t)data, (caddr_t)NULL);
      if (data->modeValue != axisData->stripMode)
        ModeButtonCallback(data->form, (caddr_t)data, (caddr_t)NULL);
    }
    if ((vd = get_vmstate(data->place, data->type)) == NULL) {
      XtSetMappedWhenManaged(data->unitLabel, False);
      XtSetMappedWhenManaged(data->vmLabel, False);
      XtSetMappedWhenManaged(data->titleLabel, False);
      XtSetMappedWhenManaged(data->titleText, False);
      XtSetMappedWhenManaged(data->scaleLabel, False);
      XtSetMappedWhenManaged(data->scaleButton, False);
```

```
    XtSetMappedWhenManaged(data->modelLabel, False);
    XtSetMappedWhenManaged(data->modelButton, False);
    XtSetMappedWhenManaged(data->unitLabel, False);
    XtSetMappedWhenManaged(data->unitText, False);
    XtSetMappedWhenManaged(data->hideButton, False);
    XtSetMappedWhenManaged(data->minLabel, False);
    XtSetMappedWhenManaged(data->minText, False);
    XtSetMappedWhenManaged(data->maxLabel, False);
    XtSetMappedWhenManaged(data->maxText, False);
    XtSetMappedWhenManaged(data->speedLabel, False);
    XtSetMappedWhenManaged(data->speedText, False);
    {
     Dimension w, h, w_r, h_r;
     Arg args[2];
     int i = 0;
     XtSetArg(args[i], XtNwidth, &w);i++;
     XtSetArg(args[i], XtNheight, &h);i++;
     XtGetValues(data->form, args, i);
     XtMakeResizeRequest(data->form, w, h+20, &w_r, &h_r);
    }
    return;
  } else {
    XtSetMappedWhenManaged(data->nameLabel, True);
    XtSetMappedWhenManaged(data->valLabel, True);
    XtSetMappedWhenManaged(data->titleLabel, True);
    XtSetMappedWhenManaged(data->titleText, True);
    XtSetMappedWhenManaged(data->unitLabel, True);
    XtSetMappedWhenManaged(data->unitText, True);
    XtSetMappedWhenManaged(data->hideButton, True);
    if (data->type == TIME) {
      XtSetMappedWhenManaged(data->minLabel, False);
      XtSetMappedWhenManaged(data->minText, False);
      XtSetMappedWhenManaged(data->maxLabel, False);
      XtSetMappedWhenManaged(data->maxText, False);
      XtSetMappedWhenManaged(data->scaleLabel, False);
      XtSetMappedWhenManaged(data->scaleButton, False);
      XtSetMappedWhenManaged(data->modeLabel, True);
      XtSetMappedWhenManaged(data->modeButton, True);
      XtSetMappedWhenManaged(data->speedLabel, True);
      XtSetMappedWhenManaged(data->speedText, True);
    } else {
      XtSetMappedWhenManaged(data->minLabel, True);
      XtSetMappedWhenManaged(data->minText, True);
      XtSetMappedWhenManaged(data->maxLabel, True);
      XtSetMappedWhenManaged(data->maxText, True);
      XtSetMappedWhenManaged(data->scaleLabel, True);
      XtSetMappedWhenManaged(data->scaleButton, True);
      XtSetMappedWhenManaged(data->modeLabel, False);
      XtSetMappedWhenManaged(data->modeButton, False);
      XtSetMappedWhenManaged(data->speedLabel, False);
      XtSetMappedWhenManaged(data->speedText, False);
    }
  }
  XtSetArg(args[0], XtNlabel, vd->name);
  XtSetValues(data->nameLabel, args, 1);
  data->unitString = vd->units;
  XtSetArg(args[0], XtNlabel, data->unitString);
  XtSetValues(data->unitText, args, 1);
}

/*
 * Dump the current data from this widget into the record specified by axisData
 * and then make the window invisible.  The widget is reused instead of being
 * destroyed and re-created for subsequent calls.  This is a little funky,
 * but no problem since there are never 2 PlotForm menus at once.
 *
 */
static void FinishAxisFormData(data, axisData)
    PFData data;
    AxisData axisData;
{
  Arg args[5];
  int i=0;
  char *str;

  XtSetArg(args[i], XtNstring, &str);i++;

  XtGetValues(data->minText, args, i);
  (void) sscanf(str, "%lf", &axisData->min);

  XtGetValues(data->maxText, args, i);
  (void) sscanf(str, "%lf", &axisData->max);

  XtGetValues(data->speedText, args, i);
  (void) sscanf(str, "%lf", &axisData->paperSpeed);

  XtGetValues(data->titleText, args, i);
  axisData->title    = XtMalloc(strlen(str)+1);
  strcpy(axisData->title, str);

  axisData->units    = data->unitsString;
  axisData->autoScale = data->scaleValue;
  axisData->stripMode = data->modeValue;
  axisData->place    = data->place;
  axisData->type     = data->type;
}

/************************************************
 *
 * The following data and code define the menus for choosing
 * all axes of a plot, and then for creating that plot.
 * These routines are presently used in ../cv/Interface.c
 *
 ************************************************/

/*
 *
 * Private data and functions for Menu
 *
 */

/*
 * struct for the menu
 */
typedef struct {
  Widget popup;
  Widget menu,  loadForm, Cancel, Create, timingGroup, timingEndGroup;
  PFData pfDataX, pfData1, pfData2, pfData3;
  AxisDataRec ad1, ad2, ad3;
#ifdef SAVEDATA
  char DispTime[80];
  Widget DispTimeText, DispTimeLabel;
```

```
#endif SAVEDATA
 Widget widget;
 void (* Callback)();
 Boolean averaging;
} *MeasData, MeasDataRec;


/* callback for cancel button, makes the menu disappear. */
/*ARGSUSED*/
static void CancelCallback(widget, closure, call)
    Widget widget;
    caddr_t closure, call;
{
 MeasData data = (MeasData) closure;

 XtPopdown(data->popup);
 FinishAxisFocusData(data->pfDataX, &data->adx);
 FinishAxisFocusData(data->pfData1, &data->ad1);
 FinishAxisFocusData(data->pfData2, &data->ad2);
 FinishAxisFocusData(data->pfData3, &data->ad3);
 FocusActive = False;
}

/* callback for creating a plot from the present state of the menu */
/*ARGSUSED*/
static void CreateCallback(widget, closure, call)
    Widget widget;
    caddr_t closure, call;
{
 MeasData data = (MeasData) closure;
 Arg args[40];
 int i;
#ifdef SAVEDATA
 double DispTimesDouble;
#endif SAVEDATA

 XtPopdown(data->popup);
 if (data->pfDataX) FinishAxisFocusData(data->pfDataX, &data->adx);
 if (data->pfData1) FinishAxisFocusData(data->pfData1, &data->ad1);
 if (data->pfData2) FinishAxisFocusData(data->pfData2, &data->ad2);
 if (data->pfData3) FinishAxisFocusData(data->pfData3, &data->ad3);
 FocusActive = False;

 /* Set up the arguments based on form data */
 i = 0;
 XtSetArg(args[i],XtNplaceX , data->adx.place);i++;
 XtSetArg(args[i],XtNtypeX  , data->adx.type);i++;
 XtSetArg(args[i],XtNlabelX , data->adx.title);i++;
 XtSetArg(args[i],XtNunitX  , data->adx.units);i++;
 XtSetArg(args[i],XtNminX   , &data->adx.min);i++;
 XtSetArg(args[i],XtNmaxX   , &data->adx.max);i++;
 XtSetArg(args[i],XtNunitsScaleX, data->adx.unitsScale);i++;
 XtSetArg(args[i],XtNstripMode, data->adx.stripMode);i++;
 XtSetArg(args[i],XtNpaperSpeed, &data->adx.paperSpeed);i++;
 XtSetArg(args[i],XtNplace1 , data->ad1.place);i++;
 XtSetArg(args[i],XtNtype1  , data->ad1.type);i++;
 XtSetArg(args[i],XtNlabel1 , data->ad1.title);i++;
 XtSetArg(args[i],XtNunit1  , data->ad1.units);i++;
 XtSetArg(args[i],XtNmin1   , &data->ad1.min);i++;
 XtSetArg(args[i],XtNmax1   , &data->ad1.max);i++;
 XtSetArg(args[i],XtNunitsScale1, data->ad1.unitsScale);i++;
 XtSetArg(args[i],XtNplace2 , data->ad2.place);i++;
 XtSetArg(args[i],XtNtype2  , data->ad2.type);i++;
 XtSetArg(args[i],XtNlabel2 , data->ad2.title);i++;
 XtSetArg(args[i],XtNunit2  , data->ad2.units);i++;
 XtSetArg(args[i],XtNmin2   , &data->ad2.min);i++;
 XtSetArg(args[i],XtNmax2   , &data->ad2.max);i++;
 XtSetArg(args[i],XtNunitsScale2, data->ad2.unitsScale);i++;
 XtSetArg(args[i],XtNplace3 , data->ad3.place);i++;
 XtSetArg(args[i],XtNtype3  , data->ad3.type);i++;
 XtSetArg(args[i],XtNlabel3 , data->ad3.title);i++;
 XtSetArg(args[i],XtNunit3  , data->ad3.units);i++;
 XtSetArg(args[i],XtNmin3   , &data->ad3.min);i++;
 XtSetArg(args[i],XtNmax3   , &data->ad3.max);i++;
 XtSetArg(args[i],XtNunitsScale3, data->ad3.unitsScale);i++;
#ifdef SAVEDATA
 (void) sscanf(data->DispTime, "%lf", &DispTimesDouble);
 XtSetArg(args[i],XtNdecayTime, &DispTimesDouble);i++;
#endif SAVEDATA
 { unsigned int timing2, timing =
     (unsigned int) XawToggleGetCurrent(data->timingGroup);
   if (timing == 15) timing = 0;
   timing2 = (unsigned int) XawToggleGetCurrent(data->timingBadGroup);
   if (timing2 == 15<<4) timing2 = 0;
   timing |= timing2;
   timing &= 0x00FF;
   if (data->averaging) timing |= 0x0800;

   XtSetArg(args[i], XtNtiming, timing);i++;
 }
 if (data->Callback) (*data->Callback)(data->widget, args, i);
}

static Pixmap savedUnder = (Pixmap) 0;
static GC imageGC = (GC) 0;
static Boolean simulating = False;

/*ARGSUSED*/
static doSaveUnder(w, client, call)
Widget w;
caddr_t client, call;
{
 if (!imageGC) {
    XGCValues xgcv;
    xgcv.subwindow_mode = IncludeInferiors;
    imageGC = XCreateGC(dpy, root, GCSubwindowMode, &xgcv);
  }
 if (savedUnder) XFreePixmap(dpy, savedUnder);

 /* POP-UP! This does a manual save-under, since I can't get the
  * automatic one to work.
  */

#ifdef DO_GRABS
 XGrabServer(dpy); /* Don't let reality change while we've got it saved */
 if (XtWindow(w))
   XGrabKeyboard(dpy, XtWindow(w), True, GrabModeAsync, GrabModeAsync, CurrentTime);
#endif DO_GRABS
 simulating = !Stopped;
 StopSimulation();
 SlowFlush();
 /* Calculate the footprint of this window. */
```

```
  {
    Dimension width=0, height=0, bdr=0;
    int x=0, y=0;
    Arg args[6];
    int i = 0;
    XtSetArg(args[i], XtNx, &x);i++;
    XtSetArg(args[i], XtNy, &y);i++;
    XtSetArg(args[i], XtNwidth, &width);i++;
    XtSetArg(args[i], XtNheight, &height);i++;
    XtSetArg(args[i], XtNborderWidth, &bdr);i++;
    XtGetValues(w, args, i);

    if (width > 100 || height > 100) {
      XSync(dpy, False); /* Save the most up-to-date picture possible. */
      savedUnder = XCreatePixmap(dpy, root, width+2*bdr, height + 2*bdr,
                                 DefaultDepth(dpy,screen));
      XCopyArea(dpy, root, savedUnder, imageGC, x, y,
                width + 2*bdr, height+ 2*bdr, 0, 0);
    } else savedUnder = 0;
  }
}

/*ARGSUSED*/
static doFixUnder(w, client, call)
Widget w;
caddr_t client, call;
{
  if (savedUnder) {
    Dimension width=0, height=0, bdr=0;
    int x=0, y=0;
    Arg args[6];
    int i = 0;
    XtSetArg(args[i], XtNx, &x);i++;
    XtSetArg(args[i], XtNy, &y);i++;
    XtSetArg(args[i], XtNwidth, &width);i++;
    XtSetArg(args[i], XtNheight, &height);i++;
    XtSetArg(args[i], XtNborderWidth, &bdr);i++;
    XtGetValues(w, args, i);
    /* POP DOWN! */
    /* Try to avoid redisplays, since we have saved all the bits. */
    XCopyArea(dpy, savedUnder, root, imageGC, 0, 0,
              width + 2*bdr, height +2*bdr, x, y);
    XFreePixmap(dpy, savedUnder); savedUnder = (Pixmap) 0;
    {/* Discard all accumulated exposure events. */
      XEvent ev;
      XSync(dpy, False);
      while (XCheckTypedEvent(dpy, Expose, &ev) == True)
          ; /* discard them since we repainted the screen instead. */
    }
#ifdef DO_GRABS
    XUngrabServer(dpy);
    XUngrabKeyboard(dpy, CurrentTime);
#endif DO_GRABS
  }
  if (simulating) ResumeSimulation();
}

/*ARGSUSED*/
static void fixButtons(w, client, call)
    Widget w;
    caddr_t client, call;
{
  Widget box = (Widget) client;
  if (/*!!!! box->core.visible && */ XtWindow(box)) {
    XtUnmapWidget(box);
    XtMapWidget(box);
  }
}

static Widget averageLabel, averageButton;
static Widget cycleLabel, continuous, isoconst, fill, iso_value, eject;
static Widget cycleToLabel, none, to_iso_c, to_fill, to_iso_r, to_eject;

/*ARGSUSED*/
static void averageCallback(w, client, call)
    Widget w;
    caddr_t client, call;
{
  MenuData data = (MenuData) client;
  Arg args[2];

  if (w) data->averaging = !data->averaging;
  XtSetArg(args[0], XtNlabel, data->averaging ? "Yes" : "No");
  XtSetValues(averageButton, args, 1);
}

static void ToButtonSetSens(n, c, ej, r, f)
    Boolean n, c, ej, r, f;
{
  XtSetSensitive(none, n);
  XtSetSensitive(to_iso_c, c);
  XtSetSensitive(to_eject, ej);
  XtSetSensitive(to_iso_r, r);
  XtSetSensitive(to_fill, f);
}

static void FromButtonSetSens(n, c, ej, r, f)
    Boolean n, c, ej, r, f;
{
  XtSetSensitive(continuous, n);
  XtSetSensitive(iso_const, c);
  XtSetSensitive(eject, ej);
  XtSetSensitive(iso_value, r);
  XtSetSensitive(fill, f);
}

/*
 * Keeps the radio buttons associated with toggle choice
 * in the right state.  Always makes the current button insensitive
 * so that it can't be inactivated by pressing again. !!!!
 * There should be a better way to do that.
 */
/*ARGSUSED*/
static void toggleCallback(w, client, call)
Widget w;
caddr_t client, call;
{
  unsigned int flags = (unsigned int) client;
  Arg args[5];
  int i=0;
  XtSetArg(args[i], XtNstate, True);i++;
  if (call) switch (flags) {
```

```
    case /* continuous: */ 15:
      XawToggleUnsetCurrent(none);
      ToButtonsSetSens(False, False, False, False, False);
      FromButtonsSetSens(False, True, True, True, True);
      XtSetValues(continuous, args, i);
      break;
    case /* iso contr: */ 4:
      XawToggleSetCurrent(none, 15<<4);
      ToButtonsSetSens(False, False, True, True, True);
      FromButtonsSetSens(True, False, True, True, True);
      XtSetValues(iso_contr, args, i);
      break;
    case /* eject: */ 2:
      XawToggleSetCurrent(none, 15<<4);
      ToButtonsSetSens(False, True, False, True, True);
      FromButtonsSetSens(True, True, False, True, True);
      XtSetValues(eject, args, i);
      break;
    case /* iso relax: */ 8:
      XawToggleSetCurrent(none, 15<<4);
      ToButtonsSetSens(False, True, True, False, True);
      FromButtonsSetSens(True, True, True, False, True);
      XtSetValues(iso_relax, args, i);
      break;
    case /* fill: */ 1:
      XawToggleSetCurrent(none, 15<<4);
      ToButtonsSetSens(False, True, True, True, False);
      FromButtonsSetSens(True, True, True, True, False);
      XtSetValues(fill, args, i);
      break;

    case /* none: */ 15<<4:
      ToButtonsSetSens(False, True, True, True, True);
      XtSetValues(none, args, i);
      break;
    case /* to_iso_c: */ 4<<4:
      ToButtonsSetSens(True, False, True, True, True);
      XtSetValues(to_iso_c, args, i);
      break;
    case /* to_eject: */ 2<<4:
      ToButtonsSetSens(True, True, False, True, True);
      XtSetValues(to_eject, args, i);
      break;
    case /* to_iso_r: */ 8<<4:
      ToButtonsSetSens(True, True, True, False, True);
      XtSetValues(to_iso_r, args, i);
      break;
    case /* to_fill: */ 1<<4:
      ToButtonsSetSens(True, True, True, True, False);
      XtSetValues(to_fill, args, i);
      break;
  }
}

/*PUBLIC*/
Widget MotePlotItems(widget, Callback, x, y, adX, ad1, ad2,ad3)
    Widget widget; /* widget to be passed to Callback (usu. parent of plot)*/
    void (* Callback)();
    int x, y; /* location for menu */
    AxisData adX, ad1, ad2, ad3;
{
    static AxisDataRec adProto = {-1, -1, "",0.0., 5, "",FALSE},
                       adTime = {0, TIME, "Time",0., 999., 5, "cm/sec",FALSE};
    static XFontStruct *cor = NULL;
    static MenuData data = NULL;
    int i, j;
    Arg args[15];

    if (FormsActive) return(NULL); /* Allow only one at a time */
    FormsActive = True;
    if (!cor) cor = XLoadQueryFont(dpy, "*-new century schoolbook-bold-r-*-*-*-180-*-*");
    if (!data) { /* only create this stuff once for speed */
      data = (MenuData) XtMalloc(sizeof(MenuDataRec));
      data->popup = XtCreatePopupShell("cvs",overrideShellWidgetClass,
#ifdef NOTDEF (ever on openwindows)||| why???
                                       XtIsComposite(widget) ? widget : XtParent(widget),
#endif NOTDEF
                                       widget,
                                       (ArgList)NULL, 0);
      data->menu = XtCreateManagedWidget("plotForm",panedWidgetClass,data->popup,
                                         (ArgList)NULL, 0);
      i=0;
      XtSetArg(args[i], XtNborderWidth, 2);i++;
      XtSetArg(args[i], XtNallowResize, True);i++;
      XtSetArg(args[i], XtNresizeToPreferred, True);i++;
      data->pdDataX = CreateAxisForm(data->menu, args, i);
      i=0;
      XtSetArg(args[i], XtNborderWidth, 2);i++;
      XtSetArg(args[i], XtNallowResize, True);i++;
      XtSetArg(args[i], XtNresizeToPreferred, True);i++;
      data->pdData1 = CreateAxisForm(data->menu, args, i);
      i=0;
      XtSetArg(args[i], XtNborderWidth, 2);i++;
      XtSetArg(args[i], XtNallowResize, True);i++;
      XtSetArg(args[i], XtNresizeToPreferred, True);i++;
      data->pdData2 = CreateAxisForm(data->menu,args, i);
      i=0;
      XtSetArg(args[i], XtNborderWidth, 2);i++;
      XtSetArg(args[i], XtNallowResize, True);i++;
      XtSetArg(args[i], XtNresizeToPreferred, True);i++;
      data->pdData3 = CreateAxisForm(data->menu, args, i);
      i=0;
      XtSetArg(args[i], XtNborderWidth, 0);i++;
      XtSetArg(args[i], XtNallowResize, False);i++;
      XtSetArg(args[i], XtNresizeToPreferred, True);i++;
      data->lastForm = XtCreateManagedWidget("lastForm", formWidgetClass,
                                             data->menu, args, i);
      XtAddCallback(data->popup, XtNpopupCallback,
                    dnSaveUnder, (caddr_t) data->menu);
      XtAddCallback(data->popup, XtNpopdownCallback,
                    dnFixUnder,(caddr_t) data->menu);
      XtAddCallback(data->popup, XtNpopdownCallback,
                    fixButtons, (caddr_t) widget);
#ifdef SAVEDATA
      i=0;
      XtSetArg(args[i], XtNlabel, "Persist (sec):");i++;
      XtSetArg(args[i], XtNborderWidth, 0);i++;
      data->DispTimeLabel = XtCreateManagedWidget("Persist", labelWidgetClass,
                                                  data->lastForm, args, i);
#endif SAVEDATA

      /* Radio group for plotting only at specific times. */
```

```
j=0;
XtSetArg(args[j], XtNtop, XtChainTop);j++;
XtSetArg(args[j], XtNbottom, XtChainTop);j++;
XtSetArg(args[j], XtNleft, XtChainLeft);j++;
XtSetArg(args[j], XtNright, XtChainLeft);j++;

i=j;
XtSetArg(args[i], XtNlabel, "Average:");i++;
XtSetArg(args[i], XtNborderWidth, 0);i++;
averageLabel = XtCreateManagedWidget("AverageLabel", labelWidgetClass,
                                     data->lastForm, args, i);
i=j;
XtSetArg(args[i], XtNlabel, "No");i++;
XtSetArg(args[i], XtNfromHoriz, averageLabel);i++;
averageButton = XtCreateManagedWidget("AverageButton", commandWidgetClass,
                                      data->lastForm, args, i);
XtAddCallback(averageButton, XtNcallback, averageCallback,
              (caddr_t)data);

i=j;
XtSetArg(args[i], XtNlabel, "Portion of cardiac cycle:");i++;
XtSetArg(args[i], XtNborderWidth, 0);i++;
XtSetArg(args[i], XtNfromVert, averageButton);i++;
cycleLabel = XtCreateManagedWidget("CycleLabel", labelWidgetClass,
                                   data->lastForm, args, i);
i=j;
XtSetArg(args[i], XtNlabel, "Continuous");i++;
XtSetArg(args[i], XtNradioData, 15);i++;
XtSetArg(args[i], XtNfromVert, cycleLabel);i++;
data->timingGroup =
  continuous = XtCreateManagedWidget("Continuous", toggleWidgetClass,
                                     data->lastForm, args, i);

i=j;
XtSetArg(args[i], XtNlabel, "Isovolumic Contraction");i++;
XtSetArg(args[i], XtNradioGroup, continuous);i++;
XtSetArg(args[i], XtNfromVert, continuous);i++;
XtSetArg(args[i], XtNradioData, 4);i++;
iso_contr = XtCreateManagedWidget("IsoC", toggleWidgetClass,
                                  data->lastForm, args, i);
i=j;
XtSetArg(args[i], XtNlabel, "Ejection");i++;
XtSetArg(args[i], XtNradioGroup, continuous);i++;
XtSetArg(args[i], XtNfromVert, iso_contr);i++;
XtSetArg(args[i], XtNradioData, 2);i++;
eject = XtCreateManagedWidget("Ej", toggleWidgetClass,
                              data->lastForm, args, i);
i=j;
XtSetArg(args[i], XtNlabel, "Isovolumic Relaxation");i++;
XtSetArg(args[i], XtNradioGroup, continuous);i++;
XtSetArg(args[i], XtNfromVert, eject); i++;
XtSetArg(args[i], XtNradioData, 8);i++;
iso_relax = XtCreateManagedWidget("IsoR", toggleWidgetClass,
                                  data->lastForm, args, i);
i=j;
XtSetArg(args[i], XtNlabel, "Filling");i++;
XtSetArg(args[i], XtNradioGroup, continuous);i++;
XtSetArg(args[i], XtNfromVert, iso_relax); i++;
XtSetArg(args[i], XtNradioData, 1);i++;
fill = XtCreateManagedWidget("Fill", toggleWidgetClass,
                             data->lastForm, args, i);
XawToggleUnsetCurrent(continuous);
XawToggleSetCurrent(continuous, (caddr_t)15);


i=j;
XtSetArg(args[i], XtNlabel, "To beginning of:");i++;
XtSetArg(args[i], XtNborderWidth, 0);i++;
XtSetArg(args[i], XtNfromHoriz, iso_contr);i++;
XtSetArg(args[i], XtNfromVert, averageButton);i++;
cycleToLabel = XtCreateManagedWidget("CycleToLabel", labelWidgetClass,
                                     data->lastForm, args, i);

i=j;
XtSetArg(args[i], XtNlabel, "1 pt.   per cycle");i++;
XtSetArg(args[i], XtNfromVert, cycleToLabel);i++;
XtSetArg(args[i], XtNfromHoriz, iso_contr);i++;
XtSetArg(args[i], XtNradioData, 15<<4);i++;
none = XtCreateManagedWidget("OnePt", toggleWidgetClass,
                             data->lastForm, args, i);
i=j;
XtSetArg(args[i], XtNlabel, "Stop at Contraction");i++;
XtSetArg(args[i], XtNfromVert, none);i++;
XtSetArg(args[i], XtNradioGroup, none);i++;
XtSetArg(args[i], XtNfromHoriz, iso_contr);i++;
XtSetArg(args[i], XtNradioData, 4<<4);i++;
data->timingEndGroup =
  to_iso_c = XtCreateManagedWidget("ToIsoC", toggleWidgetClass,
                                   data->lastForm, args, i);
i=j;
XtSetArg(args[i], XtNlabel, "Stop at Ejection");i++;
XtSetArg(args[i], XtNradioGroup, to_iso_c);i++;
XtSetArg(args[i], XtNfromVert, to_iso_c);i++;
XtSetArg(args[i], XtNfromHoriz, iso_contr);i++;
XtSetArg(args[i], XtNradioData, 2<<4);i++;
to_ejct = XtCreateManagedWidget("ToEj", toggleWidgetClass,
                                data->lastForm, args, i);
i=j;
XtSetArg(args[i], XtNlabel, "Stop at Relaxation");i++;
XtSetArg(args[i], XtNradioGroup, to_iso_c);i++;
XtSetArg(args[i], XtNfromVert, to_ejct); i++;
XtSetArg(args[i], XtNfromHoriz, iso_contr);i++;
XtSetArg(args[i], XtNradioData, 8<<4);i++;
to_iso_r = XtCreateManagedWidget("ToIsoR", toggleWidgetClass,
                                 data->lastForm, args, i);
i=j;
XtSetArg(args[i], XtNlabel, "Stop at Filling");i++;
XtSetArg(args[i], XtNradioGroup, to_iso_c);i++;
XtSetArg(args[i], XtNfromVert, to_iso_r); i++;
XtSetArg(args[i], XtNfromHoriz, iso_contr);i++;
XtSetArg(args[i], XtNradioData, 1<<4);i++;
to_fill = XtCreateManagedWidget("ToFill", toggleWidgetClass,
                                data->lastForm, args, i);

XawToggleUnsetCurrent(to_iso_c);

XtAddCallback(to_fill, XtNcallback, toggleCallback, (caddr_t)(1<<4));
XtAddCallback(to_iso_r, XtNcallback, toggleCallback, (caddr_t)(8<<4));
XtAddCallback(to_ejct, XtNcallback, toggleCallback, (caddr_t)(2<<4));
XtAddCallback(to_iso_c, XtNcallback, toggleCallback, (caddr_t) (4<<4));
XtAddCallback(none, XtNcallback, toggleCallback, (caddr_t) (15<<4));
```

```
    XtAddCallback(iso_selex, XtNcallback, toggleCallback, (caddr_t) 8);
    XtAddCallback(eject, XtNcallback, toggleCallback, (caddr_t) 2);
    XtAddCallback(iso_contr, XtNcallback, toggleCallback, (caddr_t) 4);
    XtAddCallback(continuous, XtNcallback, toggleCallback, (caddr_t) 15);
    XtAddCallback(fill, XtNcallback, toggleCallback, (caddr_t) 1);

    i=j;
    XtSetArg(args[i], XtNlabel, "Do Plot");i++;
    if (cor) { XtSetArg(args[i], XtNfont, cor);i++; }
    XtSetArg(args[i], XtNfromVert, fill); i++;
    XtSetArg(args[i], XtNborderWidth, 3);i++;
    data->Create = XtCreateManagedWidget("DoPlot", commandWidgetClass,
                                         data->lastForm, args, i);
    XtAddCallback(data->Create, XtNcallback, CreateCallback, (caddr_t)data);
    i=j;
    XtSetArg(args[i], XtNlabel, "Cancel");i++;
    if (cor) {XtSetArg(args[i], XtNfont, cor);i++; }
    XtSetArg(args[i], XtNborderWidth, 3);i++;
    XtSetArg(args[i], XtNfromVert, fill); i++;
    XtSetArg(args[i], XtNfromHoriz, data->Create);i++;
    data->Cancel = XtCreateManagedWidget("Cancel", commandWidgetClass,
                                         data->lastForm, args, i);
    XtAddCallback(data->Cancel, XtNcallback, CancelCallback, (caddr_t)data);
#ifdef SAVEDATA
    (void) strcpy(data->DispTime, "99999.9");
    i=j;
    XtSetArg(args[i], XtNwidth, 75);i++;
#ifdef XAW_BC
    XtSetArg(args[i], XtNstring, data->DispTime);i++;
    XtSetArg(args[i], XtNeditType, XawtextEdit);i++;
    XtSetArg(args[i], XtNscrollOptions, editable);i++;
#else /* ! XAW_BC */
    XtSetArg(args[i], XtNeditType, XawtextEdit);i++;
#endif /* XAW_BC */
    XtSetArg(args[i], XtNlength, 40);i++;
    XtSetArg(args[i], XtNfromHoriz, data->DispTimeLabel);i++;
    XtSetArg(args[i], XtNtranslations, parsedNonmenic);i++;
#ifdef XAW_BC
    data->DispTimeText = XtCreateManagedWidget("DispTimeText",
                         asciiStringWidgetClass, data->lastForm, args, i);
#else /* ! XAW_BC */
    data->DispTimeText = XtCreateManagedWidget("DispTimeText",
                         asciiTextWidgetClass, data->lastForm, args, i);
#endif /* XAW_BC */
    XtAddEventHandler(data->DispTimeText, EnterWindowMask, False, DoFocusKeyboard, (caddr_t)0);
#endif SAVEDATA
  }
  XtMoveWidget(data->popup, x, y);
  data->widget = widget;
  data->Callback = Callback;
  if (adX && adX->place > -1 && adX->type > -1) data->adx = *adX;
  else data->adx = adTime;
  UpdateAxisForm(data->pfDataX, "X Axis:        ", &data->adx);
  if (ad1 && ad1->place > -1 && ad1->type > -1) data->ad1 = *ad1;
  else data->ad1 = adPress;
  UpdateAxisForm(data->pfData1, "Y Axis 1:", &data->ad1);
  if (ad2 && ad2->place > -1 && ad2->type > -1) data->ad2 = *ad2;
  else data->ad2 = adPress;
  UpdateAxisForm(data->pfData2, "Y Axis 2:", &data->ad2);
  if (ad3 && ad3->place > -1 && ad3->type > -1) data->ad3 = *ad3;
  else data->ad3 = adPress;
  UpdateAxisForm(data->pfData3, "Y Axis 3:", &data->ad3);
  { unsigned int timing = 0;
    i=0;
    XtSetArg(args[i], XtNtiming, &timing);i++;
    XtGetValues(data->widget, args, i);
    if (!(timing & 0x0F)) {
      timing |= 0xFF00;
      XawToggleUnsetCurrent(data->timingEndGroup);
    } else
      XawToggleSetCurrent(data->timingEndGroup, (caddr_t) (timing & 0xF0));
    XawToggleSetCurrent(data->timingGroup, (caddr_t) (timing & 0x0F));
    toggleCallback((Widget)0, (caddr_t)timing, (caddr_t)1);
    data->averaging = timing&0x0800 ? True : False;
    averageCallback((Widget)0, (caddr_t)data, (caddr_t)0);

    XtSetArg(args[0], XtNresizeToPreferred, False);
    XtSetValues(data->lastForm, args, 1);
  }
  return(data->popup);
}
```

pfTest.c

```
#ifdef lint
static char *rcsv="$Header: /usr/hst/src/form/RCS/pfTest.c,v 2.1 90/08/10 16:55:01 dot Exp $";
#endif
/**************
 * $Source: /usr/hst/src/form/RCS/pfTest.c,v $
 * $Log:        pfTest.c,v $
 * Revision 2.1  90/08/10  16:55:01  dot
 * moved Xaw includes to X11/Xaw for R4
 *
 * Revision 2.0  90/08/06  17:53:35  tldavis
 * Final X11R3 version.
 *
 * Revision 1.4  90/06/26  10:15:52  tldavis
 * removed sys/uio dependency.
 *
 * Revision 1.3  88/10/04  08:19:54  tldavis
 * .
 *
 *
 * Revision 1.2  88/08/31  13:01:43  tldavis
 * Fully working X11 version, not fully tested.
 *
 * Revision 1.1  88/08/23  12:06:13  tldavis
 * Initial revision
 *
 **************/
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/Xaw/Command.h>
#include "../include/cvdefs.h"
#include "../plot/Plot.h"
#include "PlotForm.h"
#include <X11/Shell.h>
#include <X11/Xaw/Box.h>
#include <sys/file.h>
#include <sys/time.h>
```

```
#include <sys/types.h>
#include <fcntl.h>

#include "../event/Event.h"

Display *dpy;
Window root;
int screen;
unsigned long white_pixel, black_pixel;
static fd_set readfds, writefds, exceptfds;
static struct timeval timeout;

static Widget PLOT = NULL;
static void MakePlotCallback(parent, args, argCount)
Widget parent;
ArgList args;
int argCount;
{
  PLOT = XtCreateManagedWidget("Plot", plotWidgetClass,
                               parent, args, argCount);
}

main(argc, argv)
    int argc;
    char **argv;
{
  extern SIMULATION *getsimulation();
  Widget shell, menu, plotbox;
  int fd;

  LoadParameterData("/mit/hst/dev/data/db");
  initequvars(getsimulation(NORMAL));
  fd = open(--argc ? argv[argc] : "test.data",ORDONLY, 0);
  FD_ZERO(&readfds);
  FD_ZERO(&writefds);
  FD_ZERO(&exceptfds);
  timerclear(&timeout);

  printf("XtInitialize...\n");
  shell = XtInitialize(argv[0],"Testing", (XrmOptionDescRec *)NULL,
                       (Cardinal)0, (Cardinal *)&argc, (String *)argv);

  dpy = XtDisplay(shell);
  screen = DefaultScreen(dpy);
  root = RootWindow(dpy, screen);
  white_pixel = WhitePixel(dpy, screen);
  black_pixel = BlackPixel(dpy, screen);
/* XSynchronize(dpy, 1);*/ /* makes it slow but easier for bug tracking */

  plotbox = XtCreateManagedWidget("plotbox", boxWidgetClass, shell,(Arg *)0,0);
  printf("MakePlotMenu...\n");
  menu = MakePlotMenu(plotbox, MakePlotCallback, 300, 20,
                      (AxisData *)NULL, (AxisData *)NULL,
                      (AxisData *)NULL, (AxisData *)NULL);

  printf("XtRealizeWidgets...\n");
  XtRealizeWidget(shell);

  printf("XtPopup...\n");
  XtPopup(menu, XtGrabNone);
```

```
  {
    XEvent event;
    void DispatchData();
    for (;;) {
      while (XPending(dpy)) {
          XNextEvent(dpy, &event);
          XtDispatchEvent(&event);
      }
      if (PLOT) DispatchData(fd, PLOT);
    }
  }
}

static void DispatchData(fd, w)
int fd;
Widget w;
{
  char dataType;
  double parameter[9];
  int place, type;
  double value;

  read(fd, &dataType, sizeof(dataType));
  switch (dataType)
  {
  case PARAMETER:
    read(fd, &place, sizeof(place));
    read(fd, &type,  sizeof(type));
    read(fd, &value, sizeof(value));
    PlotModifySimulation(w, place, type, value);
    break;
  case DATA:
    if (read(fd,parameter, sizeof(parameter[0])*9)== sizeof(parameter[0])*9){
        PlotData(w, parameter);
    }
    break;
  }
}
```

C.7 Circuit diagram window

These files define the user interface for choosing parameters from a circuit diagram representation of the cardiovascular system. These routines take control of the X Window System and operate outside the normal bounds of the X Toolkit.

Circuit.h

```
/*
 * The circuit diagram is written in raw X11 code without toolbox dependancy.
 * modify_parameters() takes control of the program, returning after the
 * user has chosen some suitable parameter.
 */

extern void modify_parameters(); /* which, place, type */
/*
 * int which *//* 0: modify independant vars, 1: plot dependant vars */ /*
 * int *place, *type *//* RETURN *//*
 */
```

Xcv.h

```
/*
 *      $Source: /mit/lst/src/circuit/RCS/Xcv.h,v $
 *      $Header: /mit/lst/src/circuit/RCS/Xcv.h,v 2.0 90/08/06 17:45:26 tldavis Stab $
 */



/* default stuff for the modify-parameter window type */
#define LINEAR 0
#define LOG    1

#define MPWIN_NUM_FONT "*-courier-medium-r-*-*-*-80-*"
#define MPWIN_HDR_FONT "*-times-bold-r-*-*-*-100-*"
#define MPWIN_TIT_FONT "*-times-bold-r-*-*-*-100-*"
#define MPWIN_HEIGHT 150
#define MPWIN_WIDTH 300
#define MPWIN_TICSIZE 5
#define MPWIN_MAX_TICS 10
#define MPWIN_HEADER_HEIGHT 30
#define MPWIN_HEADER_WIDTH 50

#define DIAGRAM_FONT "*-times-bold-r-*-*-*-100-*"
#define LARGE_FONT  "*-times-bold-r-*-*-*-120-*"

#define IconWindow 0
#define CircuitWindow 1
#define ResistorWindow 2
#define CapacitorWindow 3
#define VarCapacitorWindow 4
#define DiodeWindow 5
#define WireWindow 6
#define SystemWindow 7

typedef struct circinfo_ {
  int x, y, width, height; /* specs of window in question */
  Window parent;
  int type; /* Icon, Circuit, Resistor, Capacitor, VarCapacitor, Wire etc.
               see above (always ending in Window)*/
  char *name1, *name2; /* description of window (used in icons) */
  void (*init_fn)(); /*function when passed window will fill it (or NULL)*/
  int place; /* (logical) one of 15 possibilities (see cvdefs.h) */
  int *depvars; /* list of simulated variables attached (see cvdefs.h) */
  int *indepvars; /* list of system parameters attached to this window */
} CircInfo;

/*TAG - newconfig */

typedef struct _BatchFrame{
  Window parent;
  Window self;
  Window icon;
  int x, y;
  unsigned int width, height;
  unsigned int bdrwidth;
  unsigned long border;
  unsigned long background;
}BatchFrame;
```

Xlibint.h

```
#include <X11/copyright.h>

/* $XConsortium: Xlibint.h,v 11.61 88/09/06 16:09:16 jim Exp $ */
/* Copyright 1984, 1985, 1987 Massachusetts Institute of Technology */

/*
 *      Xlibinternal.h - Header definition and support file for the internal
 *      support routines (Xlibinternal) used by the C subroutine interface
 *      library (Xlib) to the X Window System.
 *
 *      Warning, there be dragons here....
 */
#ifndef NEED_EVENTS
#define _XEVENT_
#endif

#ifdef CRAY
#ifndef _TYPES_
#define _TYPES_
#include <sys/types.h>                  /* forget to protect it... */
#endif /* _TYPES_ */
#else
#include <sys/types.h>
#endif /* CRAY */

/*
 * define the following if you want the Data macro to be a procedure instead
 */
#if defined(CRAY)
#define DataRoutineIsProcedure
#endif /* CRAY */

#include <X11/Xlib.h>
#include <X11/Xproto.h>
#include <X11/Xos.h>

#ifndef NULL
#define NULL 0
#endif
#define LOCKED 1
#define UNLOCKED 0

extern int errno;                       /* internal system error number. */
extern void bcopy();

extern (*_XIOErrorFunction)();          /* X system error reporting routine. */
extern (*_XErrorFunction)();            /* XError event reporting routine. */
extern char *_XAllocScratch();          /* fast memory allocator */
extern Visual *_XVIDtoVisual();         /* given visual id, find structure */

#ifndef BUFSIZE
#define BUFSIZE 2048                    /* X output buffer size. */
#endif
#ifndef IFBEATCH
#define IFBEATCH 8                      /* when batching, how many elements */
#endif
#ifndef CURSORFONT
```

```
#define CURSORFONT "cursor"             /* standard cursor fonts */
#endif

/*
 * X Protocol packetizing macros.
 */

/*  Need to start requests on 64 bit word boundries
 *  on a CRAY computer so add a NoOp (127) if needed.
 *  A character pointer on a CRAY computer will be zero--zero
 *  after shifting right 61 bits of it is not pointing to
 *  a word boundary.
 */
#ifdef WORD64
#define WORD64ALIGN if ((long)dpy->bufptr >> 61) {\
        dpy->last_req = dpy->bufptr;\
        *(dpy->bufptr)   = X_NoOperation;\
        *(dpy->bufptr+1) =  0;\
        *(dpy->bufptr+2) =  0;\
        *(dpy->bufptr+3) =  1;\
         dpy->request += 1;\
         dpy->bufptr += 4;\
       }
#else /* else does not require alignment on 64-bit boundaries */
#define WORD64ALIGN
#endif /* WORD64 */


/*
 * GetReq -- Get the next available X request packet in the buffer and
 * return it.
 *
 * "name" is the name of the request, e.g. CreatePixmap, OpenFont, etc.
 * "req" is the name of the request pointer.
 *
 */

#if defined(_STDC_) && !defined(UNIXCPP)
#define GetReq(name, req) \
       WORD64ALIGN\
         if ((dpy->bufptr + SIZEOF(x##name##Req)) > dpy->bufmax)\
           _XFlush(dpy);\
         req = (x##name##Req *)(dpy->last_req = dpy->bufptr);\
         req->reqType = X_##name;\
         req->length = (SIZEOF(x##name##Req))>>2;\
         dpy->bufptr += SIZEOF(x##name##Req);\
         dpy->request++

#else  /* non-ANSI C uses empty comment instead of "##" for token concatenation */
#define GetReq(name, req) \
       WORD64ALIGN\
         if ((dpy->bufptr + SIZEOF(x/**/name/**/Req)) > dpy->bufmax)\
           _XFlush(dpy);\
         req = (x/**/name/**/Req *)(dpy->last_req = dpy->bufptr);\
         req->reqType = X_/**/name;\
         req->length = (SIZEOF(x/**/name/**/Req))>>2;\
         dpy->bufptr += SIZEOF(x/**/name/**/Req);\
         dpy->request++
#endif

/* GetReqExtra is the same as GetReq, but allocates "n" additional
   bytes after the request.  "n" must be a multiple of 4!  */

#if defined(_STDC_) && !defined(UNIXCPP)
#define GetReqExtra(name, n, req) \
       WORD64ALIGN\
         if ((dpy->bufptr + SIZEOF(x##name##Req) + n) > dpy->bufmax)\
           _XFlush(dpy);\
         req = (x##name##Req *)(dpy->last_req = dpy->bufptr);\
         req->reqType = X_##name;\
         req->length = (SIZEOF(x##name##Req) + n)>>2;\
         dpy->bufptr += SIZEOF(x##name##Req) + n;\
         dpy->request++
#else
#define GetReqExtra(name, n, req) \
       WORD64ALIGN\
         if ((dpy->bufptr + SIZEOF(x/**/name/**/Req) + n) > dpy->bufmax)\
           _XFlush(dpy);\
         req = (x/**/name/**/Req *)(dpy->last_req = dpy->bufptr);\
         req->reqType = X_/**/name;\
         req->length = (SIZEOF(x/**/name/**/Req) + n)>>2;\
         dpy->bufptr += SIZEOF(x/**/name/**/Req) + n;\
         dpy->request++
#endif


/*
 * GetResReq is for those requests that have a resource ID
 * (Window, Pixmap, GContext, etc.) as their single argument.
 * "rid" is the name of the resource.
 */
#if defined(_STDC_) && !defined(UNIXCPP)
#define GetResReq(name, rid, req) \
       WORD64ALIGN\
         if ((dpy->bufptr + SIZEOF(xResourceReq)) > dpy->bufmax)\
           _XFlush(dpy);\
         req = (xResourceReq *) (dpy->last_req = dpy->bufptr);\
         req->reqType = X_##name;\
         req->length = 2;\
         req->id = (rid);\
         dpy->bufptr += SIZEOF(xResourceReq);\
         dpy->request++
#else
#define GetResReq(name, rid, req) \
       WORD64ALIGN\
         if ((dpy->bufptr + SIZEOF(xResourceReq)) > dpy->bufmax)\
           _XFlush(dpy);\
         req = (xResourceReq *) (dpy->last_req = dpy->bufptr);\
         req->reqType = X_/**/name;\
         req->length = 2;\
         req->id = (rid);\
         dpy->bufptr += SIZEOF(xResourceReq);\
         dpy->request++
#endif

/*
 * GetEmptyReq is for those requests that have no arguments
 * at all.
 */
#if defined(_STDC_) && !defined(UNIXCPP)
#define GetEmptyReq(name, req) \
```

```
        WORD64ALIGN\
          if ((dpy->bufptr + SIZEOF(xReq)) > dpy->bufmax)\
            _XFlush(dpy);\
          req = (xReq *) (dpy->last_req = dpy->bufptr);\
          req->reqType = X_##name;\
          req->length = 1;\
          dpy->bufptr += SIZEOF(xReq);\
          dpy->request++
#else
#define GetEmptyReq(name, req) \
        WORD64ALIGN\
          if ((dpy->bufptr + SIZEOF(xReq)) > dpy->bufmax)\
            _XFlush(dpy);\
          req = (xReq *) (dpy->last_req = dpy->bufptr);\
          req->reqType = X_/**/name;\
          req->length = 1;\
          dpy->bufptr += SIZEOF(xReq);\
          dpy->request++
#endif


#define SyncHandle() \
        if (dpy->synchandler) (*dpy->synchandler)(dpy)

#define FlushGC(dpy, gc) \
        if ((gc)->dirty) _XFlushGCCache((dpy), (gc))
/*
 * Data -- Place data in the buffer and pad the end to provide
 * 32 bit word alignment.  Transmit if the buffer fills.
 *
 * "dpy" is a pointer to a Display.
 * "data" is a pinter to a data buffer.
 * "len" is the length of the data buffer.
 * we can presume buffer less than 2^16 bytes, so bcopy can be used safely.
 */
#ifdef DataRoutineIsProcedure
extern void Data();
#else
#define Data(dpy, data, len) \
        if (dpy->bufptr + (len) <= dpy->bufmax) {\
                bcopy(data, dpy->bufptr, (int)len);\
                dpy->bufptr += (((len) + 3) & ~3);\
        } else\
                _XSend(dpy, data, len)
#endif /* DataRoutineIsProcedure */


/* Allocate bytes from the buffer.  No padding is done, so if
 * the length is not a multiple of 4, the caller must be
 * careful to leave the buffer aligned after sending the
 * current request.
 *
 * "type" is the type of the pointer being assigned to.
 * "ptr" is the pointer being assigned to.
 * "n" is the number of bytes to allocate.
 *
 * Example:
 *    xTextElt *elt;
 *    BufAlloc (xTextElt *, elt, nbytes)
 */
#define BufAlloc(type, ptr, n) \
    if (dpy->bufptr + (n) > dpy->bufmax) \
        _XFlush (dpy); \
    ptr = (type) dpy->bufptr; \
    dpy->bufptr += (n);

/*
 * provide emulation routines for smaller architectures
 */
#ifndef WORD64
#define Data16(dpy, data, len) Data((dpy), (char *)(data), (len))
#define Data32(dpy, data, len) Data((dpy), (char *)(data), (len))
#define _XRead16Pad(dpy, data, len) _XReadPad((dpy), (char *)(data), (len))
#define _XRead16(dpy, data, len) _XRead((dpy), (char *)(data), (len))
#define _XRead32(dpy, data, len) _XRead((dpy), (char *)(data), (len))
#endif /* not WORD64 */

#define PackData16(dpy,data,len) Data16 (dpy, data, len)
#define PackData32(dpy,data,len) Data32 (dpy, data, len)

/* Xlib manual is bogus */
#define PackData(dpy,data,len) PackData16 (dpy, data, len)

#define min(a,b) (((a) < (b)) ? (a) : (b))
#define max(a,b) (((a) > (b)) ? (a) : (b))

#define  CI_NONEXISTCHAR 0x4000 /* required because QueryFont represents
                                   a non-existant character with zero-value
                                   metrics, but requires drivers to output
                                   the default char in their place. */


#ifdef MUSTCOPY

/* a little bit of magic */
#define OneDataCard32(dpy,dstaddr,srcvar) \
  { dpy->bufptr -= 4; Data32 (dpy, (char *) &(srcvar), 4); }

#define STARTITERATE(tpvar,type,start,endcond,decr) \
  { register char *cpvar; \
  for (cpvar = (char *) start; endcond; cpvar = NEXTPTR(cpvar,type), decr) { \
    type dummy; bcopy (cpvar, (char *) &dummy, SIZEOF(type)); \
    tpvar = (type *) cpvar;
#define ENDITERATE }}

#else

/* srcvar must be a variable for large architecture version */
#define OneDataCard32(dpy,dstaddr,srcvar) \
  { *(unsigned long *)(dstaddr) = (srcvar); }

#define STARTITERATE(tpvar,type,start,endcond,decr) \
  for (tpvar = (type *) start; endcond; tpvar++, decr) {
#define ENDITERATE }

#endif /* MUSTCOPY -- used machines whose C structs don't line up with proto */
```

XDraw.c

```
/* $XConsortium: XDraw.c,v 1.7 88/09/06 16:09:31 jim Exp $ */
```

```
#include <X11/copyright.h>

/* Copyright    Massachusetts Institute of Technology    1987                */

#include "Xlibint.h"
#include <X11/Xlib.h>
#include <X11/X10.h>

#define OK_RETURN 1
#define ERR_RETURN 0
#define NULL 0

/*
Copyright 1987 by Digital Equipment Corporation, Maynard, Massachusetts,
and the Massachusetts Institute of Technology, Cambridge, Massachusetts.

                        All Rights Reserved

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in
supporting documentation, and that the names of Digital or MIT not be
used in advertising or publicity pertaining to distribution of the
software without specific, written prior permission.

DIGITAL DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL
DIGITAL BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR
ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION,
ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS
SOFTWARE.

*/

/*
 *
 * Vertices_converter: This internal routine takes a list of Vertices and
 *                     returns another list of Vertices such that the
 *                     following is true:
 *
 *      (1) No Vertex has the VertexRelative flag on
 *      (2) No Vertex has the VertexCurved flag on
 *      (3) The first Vertex has the VertexDontDraw flag on
 *      (4) The path that the return Vertex list specifies should
 *          be drawn on the screen is very close to the path that the
 *          Vertex list passed to vertices_converter specified should be
 *          drawn.  The difference is due solely to the approximation
 *          of curved segments in the original Vertex list by many
 *          small straight line segments that approximate the curve.
 *
 *  Notes:
 *
 *      (1) The first Vertex must have the VertexRelative flag
 *          turned off.  (This is not checked, just assumed)
 *      (2) VertexDontDraw is automatically turned on for the first
 *          Vertex because drawing to the first Vertex makes no sense.
 *      (3) This routine is used only by XDraw & XDrawFilled.
 *      (4) A 0 is returned iff no error occurs.
 *      (5) The passed Vertex list not changed in any way.
 *
 *  Implementation:
 *
 *      A routine from the qvss device dependent code library from the
 *      X version 10 sources written by DEC was used, abet slightly
 *      modified as it provided 99% of the desired functionality.
 *
 */

/*****************************************************************/
/*                                                               */
/*              WRITTEN BY DARYL F. HAZEL.      8509.06          */
/*                                                               */
/* Modification history:                                         */
/*                                                               */
/* Carver 8510.23  Fixed first time allocation of the coord buffer */
/*                 to allocate max(200, pathcount) amount instead of just */
/*                 200.  Potential Xserver crash problem.        */
/*                                                               */
/* Carver 8510.21  Replaced old module by module worked on by Ram Rao and */
/*                 Bob Schaffer to get better performace.        */
/*                                                               */
/* Carver 8510.03  Changed the curve line converter to return the coord */
/*                 path if a successor point does not exist for curve gen. */
/*                                                               */
/* Carver 8510.03  Fixed memory loss problem.  A coord path is only */
/*                 allocated if the current path will not fit in it and */
/*                 before the new coord path is allocated the old one is */
/*                 freed.                                        */
/*                                                               */
/* Carver 8509.24  Fixed pointer/integer mismatch {*ppathoaldnew = ...} */
/*                                                               */
/* Carlson 8509.18  Modify code to integrate into the draw curve command */
/*                                                               */
/*                 Modified by Mark Lillibridge 5/20/87 to make into */
/*                 X11 Xlib XDraw() support.                     */
/*****************************************************************/

/*
 * Definitions:
 *
 */

typedef short   WORD;
typedef short   *WORD_POINTER;

/*
 * Lookup tables, buffers used in the conversion:
 *
 */

/*
 * modified path list storage information:
 */
static Vertex *patholdptrnew = NULL;  /*pointer to current modified path list*/
static int pathlist_bytes = 0; /* size of modified path list in bytes     */
static int pathlist_segs = 0;/* size of modified path list in segments    */
static Vertex *patholdptrcoords = NULL; /*ptr to path list with coordinates */
static int path_coord_list_bytes = 0;  /* size of path list (w/coords) buff */
static int path_coord_list_segs = 0;   /* size of path list (w/coords) buff */
```

```
/*
 * THE TABLE OF PSEUDO FLOATING-POINT NUMBERS WHICH MAKE UP THE
 * MULTIPLICATIVE COEFFICIENTS FOR THE CUBIC SPLINE ALGORITHM.
 */
static short btable[] = {  0xfc18, 0x7f97, 0x0476, 0xff4e,
                           0xff8e, 0x7e5d, 0x0972, 0xff76,
                           0xf56f, 0x7c56, 0x0eec, 0xfeee,
                           0xf3ea, 0x7988, 0x14da, 0xfdc6,
                           0xf20b, 0x750a, 0x1b2f, 0xfcea,
                           0xf0e2, 0x71e1, 0x21dd, 0xfb81,
                           0xf038, 0x6ee1, 0x28d3, 0xfa16,
                           0xf001, 0x676d, 0x3000, 0xf894,
                           0xf034, 0x6176, 0x3758, 0xf709,
                           0xf0c3, 0x5b0f, 0x3eee, 0xf583,
                           0xf1a1, 0x546b, 0x4605, 0xf410,
                           0xf2c0, 0x4d41, 0x4d41, 0xf2c0,
                           0xf410, 0x4605, 0x546b, 0xf1a1,
                           0xf583, 0x3eee, 0x5b0f, 0xf0c3,
                           0xf709, 0x3758, 0x6176, 0xf034,
                           0xf894, 0x3000, 0x676d, 0xf001,
                           0xfa16, 0x28d3, 0x6ee1, 0xf038,
                           0xfb81, 0x21dd, 0x71e1, 0xf0e2,
                           0xfcea, 0x1b2f, 0x750a, 0xf20b,
                           0xfdc6, 0x14da, 0x7988, 0xf3ea,
                           0xfeee, 0x0eec, 0x7c56, 0xf56f,
                           0xff76, 0x0972, 0x7e5d, 0xff8e,
                           0xfc4e, 0x0476, 0x7f97, 0xfc18
                           };

/*
 * this is the table which determines how many straight-line segments to draw
 * for a given curved segment.
 */
static short segtable[] = {  16,  3, 56,              /* pl <= 10, 3 segments   */
                             32,  4, 40,              /* pl <= 32, 4 segments   */
                             80,  6, 24,              /* pl <= 80, 6 segments   */
                             96,  8, 16,              /* pl <= 96, 8 segments   */
                             192, 12, 8,              /* pl <= 192, 12 segments */
                             32767, 24, 00            /* pl <= 32767, 24 segments */
                           };

/*
 *
 * The Real routine:
 *
 */

static vertex convert(pathaddr, pathcount, ppathaddrnew, newpathcnt)
  Vertex  *pathaddr;            /* address of original path list          */
  int     pathcount;        /* number of points in original path list   */
  Vertex  **ppathaddr_new;  /* ptr to addr of new path list     */
  int     *newpathcnt;          /* ptr to new number of pts in path list */
{
  int p0x, p0y;                 /* coordinates of curve's predecessor point  */
  int p1x, p1y;                 /* coordinates of curve's starting point     */
  int p2x, p2y;                 /* coordinates of curve's ending point       */
  int p3x, p3y;                 /* coordinates of curve's successor point    */
  int     flag;                 /* flag word of current end-point in p.list  */
  int successor_x = 0;          /* X coordinate of curve's successor point   */
  int successor_y = 0;          /* Y coordinate of curve's successor point   */
  int little_endian;            /* stupid bmtable is little-endian */
  WORD increment;               /* btable-ptr-increment value      */
  WORD diffx, diffy;            /* absolute values of x, y end-point diffs  */
  WORD longaxis;                /* "length" of seg projected onto major axis */
  union                 /* accumulator variables for computing end-points... */
   {                    /* ...using the btable.                              */
    struct
         {
          WORD low;
          WORD high;
         } sword;
    int lword;
   } xacc, yacc;
  register Vertex *pnewpath;/* pointer for traversing modified path list */
  int     newpathcount;   /* number of end-points in modified path list*/
  register Vertex *pflag;   /* pointer used for looking for fig's last pt*/
  int     i;                /* general-purpose loop variable          */
  int     ecf_flag_set;   /* flag indicating existence of set ECF flag */
  int     count;
  int     curve_flag;           /* 0 = no curves in path; 1 = curves in path */

  /*
   * The following is necessary because of the pseudo-floating point
   * calculations done down below.
   */
  i = 1;
  if (*(char *) &i)
    little_endian = 1;
  else
    little_endian = 0;

  {
   register Vertex *poldpath;/* pointer for traversing original path list */
   register Vertex *p_coord_path; /* ptr to path list with coordinates    */

   /* Init the return path count to 0 in case of error */
   *newpathcnt = 0;
   newpathcount = 0;

   /* Use path buffers from last time */
   pnewpath = pathaddr_new;
   p_coord_path = pathaddr_coords;

   /*
    * if size of coordinate path list buffer is not at least large enough to
    * accommodate the specified path list, allocate enough memory to do so
    */
   if (path_coord_list_segs == 0)                    /* first time through*/
    {
        path_coord_list_segs = max (200, pathcount);
        path_coord_list_bytes = path_coord_list_segs * sizeof(Vertex);
        if ((pathaddr_coords = (Vertex *)malloc(path_coord_list_bytes)) != 0)
          p_coord_path = pathaddr_coords;
        else
          return(ERR_RETURN);
    }
   else                                          /* all but first time through    */
    {
        /* If coord buffer is large enough for this path then reuse it, other
           wise free the old one and get a new one (Carver) 8510.03 */
        if (path_coord_list_segs < pathcount)
```

```
        {
          free( pathaddr_coords);
          path_coord_list_segs = pathcount;
          path_coord_list_bytes = path_coord_list_segs * sizeof(Vertex);
          if ((pathaddr_coords =(Vertex *)malloc(path_coord_list_bytes)) !=0)
            p_coord_path = pathaddr_coords;
          else
            return(ERR_RETURN);
        };
    }

    /*
     * Copy given Vertex list (pathaddr) to coordinate buffer converting
     * relative coordinates to absolute coordinates.
     *
     * Sets curve_flag iff at least one Vertex has the VertexCurved flag set.
     *
     */

    poldpath = pathaddr;
    curve_flag = 0;

    *p_coord_path = *poldpath++;        /* First Vertex is a special case */
    p_coord_path->flags = (p_coord_path->flags | VertexDontDraw)
                        & ~VertexRelative;
    if ((p_coord_path++)->flags & VertexCurved)
      curve_flag = 1;
    for (i = pathcount - 1; i > 0; i--)
      {
        if (poldpath->flags & VertexRelative)
          {                             /* compute coordinates using last pt */
            p_coord_path->x = poldpath->x + p_coord_path[-1].x;
            p_coord_path->y = poldpath->y + p_coord_path[-1].y;
            p_coord_path->flags = poldpath->flags & ~VertexRelative;
          }
        else *p_coord_path = *poldpath;
        p_coord_path++;
        if ((poldpath++)->flags & VertexCurved)
          curve_flag = 1;
      }
    }       /* End of scope for poldaddr, p_coord_path */

    /*
     * if it has been determined that there are no curved segments and points in
     * the specified path list, return the path list containing the coordinates
     * to the calling routine.
     */
    if (!curve_flag)
      {
        *newpathcnt    = pathcount;
        *ppathaddr_new = pathaddr_coords;
        return(OK_RETURN);
      }

    /*
     * if size of modified path list buffer is not at least large enough to
     * accommodate the specified path list allocate enough memory to do so
     */
    pathaddr = pathaddr_coords;
    if (pathlist_segs < pathcount)
      {
        if (pathlist_segs == 0)         /* Fix storage leak --- MDL 5/20/87 */
            free( pathaddr_new );
        pathlist_segs = pathcount;
        pathlist_bytes = pathcount * sizeof(Vertex);
        if ((pathaddr_new = (Vertex *)malloc(pathlist_bytes)) != 0)
            pnewpath = pathaddr_new;
        else
            return(ERR_RETURN);
      }

    /*
     * initialize the beginning and ending coordinates of the first segment
     */
    p1x = 0;
    p1y = 0;
    p2x = 0;
    p2y = 0;

    {
      register WORD *ptmtable;      /* table of multiplicative coeffics.*/
      register WORD *psegtable;     /* table used to determ num_sub-segs*/
      register WORD ns;             /* num segments into which curve is divided */

      /*
       * MAIN LOOP OF THE PATH_LIST_CONVERTER ROUTINE
       */
      for ( count=pathcount ; count > 0; count--)
        {
          p0x = p1x;    /* save previous values of path-list coordinate pairs*/
          p0y = p1y;
          p1x = p2x;
          p1y = p2y;

          /*
           * read next end-point's coordinates from the path list
           */
          p2x = pathaddr->x;
          p2y = pathaddr->y;
          flag = (pathaddr++)->flags;

          /*
           * CURVED-segment considerations
           */
          if (flag & VertexCurved)
            {
              /*
               * determine which point to use as the successor point:  the next
               * point in the list (if there is one), or a previously-saved point
               * (when drawing closed figures)
               */
              if (flag & VertexEndClosed)       /* last segment of closed fig*/
                {
                  p3x = successor_x;
                  p3y = successor_y;
                }
              else                              /* stand-alone curved segment*/
                {
                  if (count <= 1)               /* no points to use as successor pt */
                    {

                      /* in this case draw the coord path */
```

```
        *newpathcnt    = pathcount;
        *ppathaddr_new = pathaddr_coords;
        return(OK_RETURN);
      }
    else
      {
       /*
        * read next end-point's coordinates from the path list...
        */
       p3x = pathaddr->x;
       p3y = pathaddr->y;

       if (flag & VectorStartClosed)
         {    /* first segment of closed figure */
              /*
               * save P3 as successor pt for closed-fig-ending curve
               */
              successor_x = p3x;
              successor_y = p3y;

              /*
               * Traverse the original path list looking for ECF flag
               * After finding it, back pointer up to previous point
               * and save the information for use as curve's pred. pt
               */
              ecf_flag_set = 0;
              pflag = pathaddr;
              for (i = count; i > 0; i--)
               {
                if (pflag->flags & VectorEndClosed)
                  {
                      pflag--;
                      p1x = pflag->x;
                      p1y = pflag->y;
                      ecf_flag_set = 1;
                      break;
                  }
                pflag++;
               }
              if (!ecf_flag_set)
               return(ERR_RETURN);
         }    /* end code pertaining to starting figure*/
      }       /* end code pertaining to successor points  */
}             /* end code pertaining to stand-alone curved segs*/

if ((flag & VectorDontDraw) == 0)
 {
     /*
      * determine the "length" of the segment along the major axis
      */
     if ((diffx = p2x - p1x) < 0)
       diffx = -diffx + 1;
     if ((diffy = p2y - p1y) < 0)
       diffy = -diffy + 1;
     lmajoraxis = (diffx >= diffy) ? diffx : diffy;
     if (lmajoraxis == 0)          /* for vector to have length of...  */
       lmajoraxis = 1;             /* ...at least one                  */

     /*
      * compute M, the number of sub-segments into which a curved
      * segment is divided
      */
     pseptable = segtable;
     while (lmajoraxis > *pseptable++)/* search for appropriate..*/
       pseptable += 2;                /* ...table entry          */
     m = *pseptable++;                /* read number of segments */
     increment = *pseptable >> 1;

     /*
      * determine if there is enough room remaining in the modified
      * path-list buffer to hold ALL of the curve's sub-segment in-
      * formation; if not, double the buffer size (if possible).
      */
     while ((newpathcount + m) > pathlist_segs)
      {
        pathlist_segs *= 2;
        pathlist_bytes *= 2;
        if ((pathaddr_new = (Vertex *)realloc(pathaddr_new,
                                              pathlist_bytes)) != 0)
          pnewpath = pathaddr_new + newpathcount;
        else
          return(ERR_RETURN);
      }

     /*
      * generate end-points of sub-segs into which curve is divided
      */
     ptantable = tantable; /* initialize the pointer to the En table*/
     for ( ; m > 1; m--)
      {
        ptantable += increment;
        xxa.lword = *ptantable * p0x;
        yya.lword = *ptantable++ * p0y;
        xxa.lword += *ptantable * p1x;
        yya.lword += *ptantable++ * p1y;
        xxa.lword += *ptantable * p2x;
        yya.lword += *ptantable++ * p2y;
        xxa.lword += *ptantable * p3x;
        yya.lword += *ptantable++ * p3y;
        xxa.lword = xxa.lword << 1;     /* double values...  */
        yya.lword = yya.lword << 1;     /* ...bc accts for it*/
        xxa.lword += 0x8000; /* round off the accum value */
        yya.lword += 0x8000;
        if (!lists_endian) {
             pnewpath->x = xxa.sword.high;/* the X coordinate  */
             pnewpath->y = yya.sword.high;/* the Y coordinate  */
        } else {
             pnewpath->x = xxa.sword.low;/* the X coordinate  */
             pnewpath->y = yya.sword.low;/* the Y coordinate  */
        }
        (pnewpath++)->flags = 0;        /* the flag word     */
        newpathcount++;/* increment segment end-point count*/
       }                /* end sub-segment end-point compute loop*/
   }            /* end DRAW-segment considerations code      */
 }              /* end CURVED-segment considerations code    */

/*
 * put end-point from original path list into modified list
 */
pnewpath->x = p2x;
pnewpath->y = p2y;
(pnewpath++)->flags = flag & ~VectorCurved;
```

```
        newpathcount++;

        /*
         * determine whether or not the modified path list is full, and if so,
         * double its size
         */
        if (newpathcount == pathlist_segs)
          {
            pathlist_bytes *= 2;
            pathlist_segs *= 2;
            if ((pathaddr_new = (Vertex *)realloc(pathaddr_new,
                                                  pathlist_bytes)) != 0)
              newpath = pathaddr_new + newpathcount;
            else
              return(ERR_RETURN);
          }
      }             /* END of PATH_LIST_CONVERTER's main loop       */

    /*
     * return the address of the modified path list and the number of segment
     * and--points it contains
     */
    *newpathcnt    = newpathcount;
    *ppathaddr_new = pathaddr_new;
  }

  return(OK_RETURN);
}

/* Written by Mark Lillibridge */

static XPoint *XDraw_points = NULL;   /* Buffer to hold list of points for */
static int    point_count = 0;        /* use in calling XDrawLines         */

Status XDraw (dpy, d, gc, vlist, vcount)
    register Display *dpy;
    Drawable d;
    GC gc;
    Vertex *vlist;
    int vcount;
{
  Vertex *newvlist;
  int newvcount;
  XPoint *p;
  int pcount;

  /* If less than 2 vertices, we don't have to do anything (no lines) */
  if (vcount<2)
    return(OK_RETURN);

  /* Convert curved lines to straight lines & change relative references to */
  /* absolute references. */
  if (!vertices_convert( vlist, vcount, &newvlist, &newvcount))
    return(ERR_RETURN);

  /* Insure we have enough buffer space */
  if (point_count < newvcount) {
    if (point_count != 0)
      free( XDraw_points );
    if (!(XDraw_points = (XPoint *) malloc( newvcount * sizeof(XPoint) )))
      return(ERR_RETURN);
    point_count = newvcount;
  }

  /*
   * Draw the lines defined by newvlist using separate XDrawLines calls
   * to insure that all the lines that should be joined are and that closed
   * figures are joined correctly.
   */
  while (newvcount>0) {
    p = XDraw_points;           /* Put points in buffer */
    pcount = 0;
    p->x = newvlist->x;         /* Copy first point */
    (p++)->y = (newvlist++)->y;
    newvcount--; pcount++;

    /*
     * Copy more points until out of points or need to stop XDrawLine
     * because either we don't want to join to the next point
     * (VertexDontDraw) or we want to stop after the next point so that
     * a closed figure will be joined correctly.  (We have to stop before
     * a VertexStartClosed because the vertex marked VertexStartClosed
     * must be the first vertex in its XDrawLines call to get joining
     * to work correctly.
     */
    while (newvcount > 0 && !(newvlist->flags & (VertexDontDraw |
                                                 VertexStartClosed |
                                                 VertexEndClosed))) {
      p->x = newvlist->x;
      (p++)->y = (newvlist++)->y;
      newvcount--; pcount++;
    }

    /*
     * If stopped only because of need to start a new XDrawLines, copy
     * next point but don't advance pointer so two XdrawLines act like one.
     */
    if ( newvcount > 0 && !(newvlist->flags & VertexDontDraw) ) {
      p->x = newvlist->x;
      (p++)->y = newvlist->y;
      pcount++;
    }

    /* Do the XDrawLines if there are any lines to draw */
    if (pcount>1)
      XDrawLines(dpy, d, gc, XDraw_points, pcount, CoordModeOrigin);
  }

  return(OK_RETURN);
}

Status XDrawFilled (dpy, d, gc, vlist, vcount)
    register Display *dpy;
    Drawable d;
    GC gc;
    Vertex *vlist;
    int vcount;
{
  Vertex *newvlist;
  int newvcount;
  XPoint *p;
  int pcount;
```

```
/* If less than 2 vertices, we don't have to do anything (no lines) */
if (vcount<2)
  return(OK_RETURN);

/* Convert curved lines to straight lines & change relative references to */
/* absolute references. */
if (!vertices_converter( vlist, vcount, &newvlist, &newvcount))
  return(ERR_RETURN);

/* Insure we have enough buffer space */
if (point_count < newvcount) {
  if (point_count != 0)
    free( XDraw_points );
  if (!(XDraw_points = (XPoint *) malloc( newvcount * sizeof(XPoint) )))
    return(ERR_RETURN);
  point_count = newvcount;
}

/*
 * Draw the lines defined by newvlist using seperate XDrawLines calls
 * to insure that all the lines that should be joined are and that closed
 * figures are joined correctly.
 */
while (newvcount>0) {
  p = XDraw_points;             /* Put points in buffer */
  pcount = 0;
  p->x = newvlist->x;           /* Copy first point */
  (p++)->y = (newvlist++)->y;
  newvcount--; pcount++;

  /*
   * Copy more points until out of points or need to stop XDrawLine
   * because either we don't want to join to the next point
   * (VertexDontDraw) or we want to stop after the next point so that
   * a closed figure will be joined correctly. (We have to stop before
   * a VertexStartClosed because the vertex marked VertexStartClosed
   * must be the first vertex in its XDrawLines call to get joining
   * to work correctly.
   */
  while (newvcount > 0 && !(newvlist->flags & (VertexStartClosed |
                                               VertexEndClosed))) {
    p->x = newvlist->x;
    (p++)->y = (newvlist++)->y;
    newvcount--; pcount++;
  }

  /*
   * If stopped only because of need to start a new XDrawLines, copy
   * next point but don't advance pointer so two XdrawLines act like one.
   */
  if ( newvcount > 0 ) {
    p->x = newvlist->x;
    (p++)->y = newvlist->y;
    pcount++;
  }

  /* Do the XDrawLines if there are any lines to draw */
  if (pcount>1) {
        XFillPolygon(dpy, d, gc, XDraw_points, pcount, Complex,
                     CoordModeOrigin);
    }
  }

  return(OK_RETURN);
}
```

diagfns.c

```
#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/X10.h>
#include <math.h>
#include "Xcvh"
extern XContext circuit_data; /* from diagram.c */
extern GC mainGC; /* from diagram.c */
extern Display *dpy;

Vertex vlist[50];
XSegment slist[50];
int addv_next, addh_next;

draw_capacitor(w, x0, y0, x1, y1, width)/* ANY screen orientation*/
Window w;
int x0, y0; /* start coordinates (connection site) */
int x1, y1; /* end of ground symbol */
int width; /* width of parallel lines */
{
  int dx, dy;
  double length;

  dx = x1-x0; dy = y1-y0;
  length = sqrt((double)(dx*dx + dy*dy));

  addv_next = 0;
  addv(x0, y0, VertexDontDraw); /*location of connection */
  addv((int)(dx*2./5.),(int)(dy*2./5.), VertexRelative);
  /* connecting wire segment */
  addv((int)(-width/2.*dy/length),(int)(width/2.*dx/length),
       VertexRelative | VertexDontDraw); /*to end of plate*/
  addv((int)(width*dy/length),(int)(-width*dx/length),
       VertexRelative );/*1st*/
  addv(dx/10, dy/10, VertexRelative | VertexDontDraw);/*to next plate end*/
  addv((int)(-width*dy/length),(int)(width*dx/length),
       VertexRelative );/*2nd*/
  addv((int)(x0+5./10.*dx),(int)(y0+5./10.*dy), VertexDontDraw);
  /*mid of plate end*/
  addv((int)(x0+4.5/5.*dx),(int)(y0+4.5/5.*dy), 0);
  /*draw wire from cap to ground*/
  addv((int)(-width/4.*dy/length),(int)(width/4.*dx/length),
       VertexRelative | VertexDontDraw);/*pos to end of big ground plate*/
  addv((int)(width/2.*dy/length),(int)(-width/2.*dx/length),
       VertexRelative ); /*1st ground plate*/
  addv((int)(x0+4.75/5.*dx),(int)(y0+4.75/5.*dy),VertexDontDraw);
  addv((int)(-width/8.*dy/length),(int)(width/8.*dx/length),
       VertexRelative | VertexDontDraw);
  addv((int)(width/4.*dy/length),(int)(-width/4.*dx/length),
       VertexRelative ); /*2nd ground plate*/
  addv(x1, y1, VertexDontDraw);
  addv((int)(-width/16.*dy/length),(int)(width/16.*dx/length),
       VertexRelative | VertexDontDraw);
```

```
  addv((int)(width/8.*dy/length),(int)(-width/8.*dx/length), VertexRelative
    );/*last ground plate*/
  XDraw(dpy, w, mainGC, vlist, addv_next);
}

draw_diodes(w, x0, y0, x1, y1, width) /* up or down ONLY! (x1==x0) */
Window w;
int x0, y0; /* inflow connection site */
int x1, y1; /* outflow connection site */
{
  if (x0 != x1) return; /* Can only draw vertical diodes */
  addv_next=0;
  addv((int)(x1-width/2), y1, VertexDontDraw);
  addv((int)(x1+width/2), y1, 0);
  addv(x1, y1, VertexDontDraw);
  addv((int)(x1+width/2), y0, 0);
  addv((int)(x1-width/2), y0, 0);
  addv(x1, y1, 0);
  XDraw(dpy, w, mainGC, vlist, addv_next);
/*NEW ... not quite right.
  adds_next=0;
  adds((int)(x1-width/2), y1, (int)(x1+width/2), y1);
  adds(x1, y1, (int)(x1+width/2), y0);
  adds((int)(x1-width/2), y0, x1, y1);
  XDrawSegments(dpy, w, mainGC, slist, adds_next);
*/
}

draw_resistor(w, x0, y0, x1, y1, width)/* up->down or left->right only*/
Window w;
int x0, y0;
int x1, y1;
{
  int delta;

  addv_next = 0;
  addv(x0, y0, VertexDontDraw);
  if (y0==y1) {
    delta = (x1-x0)/6;
    addv((int)(delta/2), (int)(-width/2), VertexRelative);
    addv(delta, width, VertexRelative);
    addv(delta, -width, VertexRelative);
    addv(delta, width, VertexRelative);
    addv(delta, -width, VertexRelative);
    addv(delta, width, VertexRelative);
    addv(x1, y1, 0);
  } else {
    delta = (y1-y0)/6;
    addv((int)(-width/2), (int)(delta/2), VertexRelative);
    addv(width, delta, VertexRelative);
    addv( -width, delta, VertexRelative);
    addv(width, delta, VertexRelative);
    addv( -width, delta, VertexRelative);
    addv(width, delta, VertexRelative);
    addv(x1, y1, 0);
  }
  XDraw(dpy, w, mainGC, vlist, addv_next);
/*NEW ... not quite right
  adds_next = 0;
  if (y0==y1) {
    delta = (x1-x0)/6;
    adds(x0, y0, (int)(delta/2), (int)(-width/2));
    adds(delta, width, delta, -width);
    adds(delta, width, x1, y1);
  } else {
    delta = (y1-y0)/6;
    adds(x0, y0, (int)(-width/2), (int)(delta/2));
    adds(width, delta, -width, delta);
    adds(width, delta, x1, y1);
  }
  XDrawSegments(dpy, w, mainGC, slist, adds_next);
*/
}

addv(x, y, flags)
int x, y, flags;
{
  vlist[addv_next].x = x;
  vlist[addv_next].y = y;
  vlist[addv_next++].flags = flags;
}

adds(x1, y1, x2, y2)
int x1, y1, x2, y2;
{
  slist[adds_next].x1 = x1;
  slist[adds_next].y1 = y1;
  slist[adds_next].x2 = x2;
  slist[adds_next++].y2 = y2;
}

mk_heart(w, x0, y0, x1, y1, ym, rev, heartMap)
Window w;
int x0, y0, x1, y1, ym, rev;
Pixmap heartMap;
{
  int dx, dy, xmid, tmp;
  CircInfo *cinfo;

  if (!heartMap) return;
  if (IXFindContext(dpy, w, circuit_data, (caddr_t *)&cinfo)) {
    dx = x1-x0;
    dy = y1-y0;

    XCopyArea(dpy, heartMap, w, mainGC,0,0, cinfo->width, cinfo->height, 0, 0);
    switch (cinfo->type) {
    case IconWindow:
      mk_icn(w);
      break;
    case CircuitWindow:
      if (rev) {tmp = x0; x0 = x1; x1 = tmp;}
      rev = rev ? -1 : 1;
      xmid = x0+rev*(int)(0.5*dx);
      dy = ym-y0;
      if (rev == -1) {tmp = ym; ym = y0; y0 = tmp;}
      XDrawLine(dpy, w, mainGC, xmid, y0, xmid, (int)(y0+rev*dy/5));
      draw_diodes(w, xmid, (int)(y0+rev*dy/5),
                  xmid, (int)(y0+rev*dy/5+rev*dy/15), (int)(dy/12));
      XDrawLine(dpy, w, mainGC, xmid, (int)(y0+rev*dy/5+rev*dy/15),
                xmid, (int)(y0+2*rev*dy/5));
      draw_diodes(w, xmid, (int)(y0+rev*2*dy/5),
                  xmid, (int)(y0+rev*2*dy/5+rev*dy/15), (int)(dy/12));
```

```
    XDrawLine(dpy, w, mainGC, xmid, (int)(y0+rev*2*dy/5+rev*dy/15),
                  xmid, ym-rev*20-rev*15);
    draw_resistor(w, xmid, ym-rev*20-rev*15, xmid, ym-rev*20, 8);
    XDrawLine(dpy, w, mainGC, xmid, ym-rev*20, xmid, ym);
    tmp = y0+rev*dy/5+rev*dy/10+rev*dy/30;
    draw_capacitor(w, xmid, tmp, x0+rev*5, tmp, (int)(dy/10));
    break;
  }
 }
}
```

diagram.c

```
/*
 * diagram.c
 */

static short x1, x2, x3, x4, x5, x6, x7, x8, x9, x10,
        x11, x12, x13, x14; /*x points*/
static short yy1, y2, y3, y4, y5, y6, y7, y8, y9, yy10, ym; /*y points */
static int dx, dy; /* heart window width, height */

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/X10.h>
#include <X11/cursorfont.h>
#include <stdio.h>
#include <strings.h>
#include "Xcv.h"
#include "../sim/CVDefs.h"
#include "../data/Parameter.h" /* for get_varlist() */
extern int addv_next, addi_next;
VarData *vars[15][6];
extern Vertex vlist[]; /* for using XDraw Vertex data structure */
extern XSegment slist[]; /* for using XDrawSegments XSegment data structure */
extern Display *dpy;
extern int screen;
extern Window root;
extern unsigned long white_pixel, black_pixel;
extern GC mainGC, diagGC, invertGC;
extern Window buttons; /* from xcvf.par.c */
extern int examp_type; /* from xcvf.par.c */
extern void choose_variable(); /* data */
extern char *malloc();

/*PUBLIC*/
XContext circuit_data = NULL;/* global assoc between curcuit subwins and data*/
XContext assoc_win = NULL; /* the window's associated "icon" window */
GC mainGC, diagGC, invertGC;
XFontStruct *diagram_font = NULL; /* font used for filling icons */
Cursor left_ptr_cursor; /* cursor to be used throughout system */
static BoxInPixmap dadi[40], icondadi[11], compdadi[16]; /* making subwindows */
static int dadi_next, icondadi_next, dadi_end, icondadi_end, compdadi_next;
CircInfo *(dadi_info[40]), *(icondadi_info[11]), *(compdadi_info[16]);
    /*data for each window*/
static Pixmap HiBMap = NULL, LiBMap = NULL;

void CreateWindowBatch(list, count)
BoxInPixmap *list;
int count;
{
 register int n;
 for (n=0; n<count; n++) {
  list[n].self = XCreateSimpleWindow(dpy, list[n].parent, list[n].x,
                list[n].y, list[n].width, list[n].height, list[n].bdrwidth,
                list[n].border, list[n].background);
  list[n].icon = list[n].self;
 }
}

/*PUBLIC*/ void mk_icn(w)
Window w;
{
 CircInfo *cinfo;
 int x0, x1, y0, y1;
 short fheight;

 if (!XFindContext(dpy, w, circuit_data, (caddr_t *)&cinfo)) {
  if (diagram_font)
   fheight = diagram_font -> max_bounds.ascent +
        diagram_font -> max_bounds.descent;
  else fheight = 10;
  y0 = (cinfo->height - 3*fheight)/2;
  y1 = y0 + 2*fheight;
  if (diagram_font) {
   x0 = (cinfo->width - XTextWidth(diagram_font, cinfo->name1,
                                   strlen(cinfo->name1)))/2;
   x1 = (cinfo->width - XTextWidth(diagram_font, cinfo->name2,
                                   strlen(cinfo->name2)))/2;
  } else
   x0 = x1 = (cinfo->width - 6)/2;
  XDrawString(dpy, w, mainGC, x0, y0, cinfo->name1,
              strlen(cinfo->name1));
  XDrawString(dpy, w, mainGC, x1, y1, cinfo->name2,
              strlen(cinfo->name2));
 }
}

static void mk_lv(w)
Window w;
{
 CircInfo *cinfo;
 if (!XFindContext(dpy, w, circuit_data, (caddr_t *)&cinfo)) {
  mk_lsym(w, x1-cinfo->x, y6+3-cinfo->y,
          x12-2-cinfo->x, y6-cinfo->y, ym-cinfo->y, 0, LiBMap);
  XDrawLine(dpy, w, diagGC, x13-cinfo->x, 0, x13-cinfo->x, 3);
  XDrawLine(dpy, w, diagGC, x13-cinfo->x, ym-cinfo->y,
            x13-cinfo->x, cinfo->height);
 }
}

static void mk_ast(w)
Window w;
{
 XDrawLine(dpy, w, mainGC, 0, 50, 80, 50);
 draw_capacitor(w, 40, 50, 40, 90, 14);
}

static void mk_cap(w)
Window w;
{
 XDrawLine(dpy, w, mainGC, 0, 50, 45, 50);
```

```
  draw_resistor(w, 45, 50, 65, 50, 8);
  XDrawLine(dpy, w, mainGC, 65, 50, 110, 50);
}

static void mk_ov(w)
Window w;
{
  XDrawLine(dpy, w, mainGC, 0, 50, 45, 50);
  draw_capacitor(w, 20, 50, 20, 90, 15);
  draw_resistor(w, 45, 50, 65, 50, 8);
  XDrawLine(dpy, w, mainGC, 65, 50, 80, 50);
}

static void mk_rv(w)
Window w;
{
  CircInfo *cinfo;
  if (IXFindContext(dpy, w, circuit_data, (caddr_t *)&cinfo)) {
    mk_hvar(w, x11-cinfo->x, y6+3-cinfo->y,
            x10-2-cinfo->x, y8-cinfo->y, yss-cinfo->y, 1, RHMap);
    XDrawLine(dpy, w, diagGC, x14-cinfo->x, 0, x14-cinfo->x, 3);
    XDrawLine(dpy, w, diagGC, x14-cinfo->x, yss-cinfo->y, x14-cinfo->x,
              cinfo->height);
  }
}

static void mk_pv(w)
Window w;
{
  XDrawLine(dpy, w, mainGC, 0, 50, 15, 50);
  draw_resistor(w, 15, 50, 35, 50, 8);
  XDrawLine(dpy, w, mainGC, 35, 50, 80, 50);
  draw_capacitor(w, 60, 50, 60, 90, 15);
}

/*ARGSUSED*/
static void mk_ank(w)
Window w;
{
;/*noop*/
}

static void mk_sys(w)
Window w;
{

  Window icon;
  static CircInfo *cinfo = NULL;

  if (!cinfo) {
    cinfo = (CircInfo *) malloc(sizeof(CircInfo));
    cinfo->type = SystemWindow;
    cinfo->mk_fn = NULL;
    cinfo->place = SYSTEM;
    cinfo->depvars = get_vardict(SYSTEM, SYSTEMOR, DEPENDENT);
    cinfo->indepvars = get_vardict(SYSTEM, SYSTEMOR, INDEPENDENT);
  }
  if (IXFindContext(dpy, w, assoc_win, (caddr_t *)&icon)) {
    XMapWindow(dpy, icon);
    XUnmapWindow(dpy, w);
  }
```

```
1340    choose_variable(cinfo, mousig_type);
}

/*PUBLIC*/ void diagram_map_icons()
{
register int n;

for (n=0; n<iconinfo_cnt; n++)
  XMapWindow(dpy, iconinfo[n].self);
for (n=iconinfo_cnt; n<info_cnt; n++)                                    1410
1350  XMapWindow(dpy, info[n].self);
}

static char *mcpy(s1, s2, n) /* my strncpy which copies right through nulls */
char *s1, *s2;
int n;
{
while (n--) *(s1+n) = *(s2+n);
return(s1);
}                                                                        1420
1360
static void icon_define(info, icon_fn)
CircInfo *info;
void (*icon_fn)();
{
  CircInfo *iconinfo;

  mcpy((char *)&(iconinfo[iconinfo_next]), (char *)&(info[iconinfo_next]),
       sizeof(iconinfo[0]));
  iconinfo = (CircInfo *)malloc(sizeof(CircInfo));                       1430
1370  mcpy((char *) iconinfo, (char *) info, sizeof(CircInfo));
  iconinfo->mk_fn = icon_fn;
  iconinfo->type = IconWindow;
  iconinfo_info[iconinfo_next++] = iconinfo;
}

static void bestguess(parent, x1, y1, x2, y2, name1, name2, mkfn, place)
Window parent;
int x1, y1, x2, y2;
char *name1, *name2;                                                     1440
1380  void (*mk_fn)();
int place;
{
  CircInfo *info;

  info = (CircInfo *) malloc(sizeof(CircInfo));
  info[info_next].parent    = info->parent    = parent;
  info[info_next].x         = info->x         = x1;
  info[info_next].y         = info->y         = y1;
  info[info_next].width     = info->width     = x2-x1;                   1450
1390  info[info_next].height    = info->height    = y2-y1;
  info[info_next].bdrwidth  = 2;
  info[info_next].border    = white_pixel;
  info[info_next].background = black_pixel;
  info->type   = CircuitWindow;
  info->place  = place;
  info->name1  = name1;
  info->name2  = name2;
  info->mk_fn  = mk_fn;
  info->depvars = NULL;                                                  1460
1400  info->indepvars = NULL;
```

```
  defs_info[defs_next++] = info;
  icon_define(info, mk_icon);
}

static void win(parent, x1, y1, x2, y2, name1, name2, width, place)
Window parent;
int x1, y1, x2, y2;
char *name1, *name2;
void (*mk_fn)();
int place;
{
  box(parent,x1,y1,x2,y2,name1,name2,width,place);
  defs[defs_next-1].bdrwidth = 0;
  icondefs[icondefs_next-1].bdrwidth = 0;
  icondefs_info[defs_next-1]->mk_fn = mk_fn;
  return;
}

/* THESE ARE FOR INPUTONLY WINDOWS! */
static void comp(parent, x1, y1, x2, y2, place, component, windowtype)
Window parent;
int x1, y1, x2, y2;
int place, component, windowtype;
{
  CircInfo *info;

  info = (CircInfo *) malloc(sizeof(CircInfo));
  compdefs[compdefs_next].parent     = info->parent     = parent;
  compdefs[compdefs_next].x          = info->x          = x1;
  compdefs[compdefs_next].y          = info->y          = y1;
  compdefs[compdefs_next].width      = info->width      = x2-x1;
  compdefs[compdefs_next].height     = info->height     = y2-y1;
  compdefs[compdefs_next].bdrwidth   = 0;
  compdefs[compdefs_next].border     = white_pixel;
  compdefs[compdefs_next].background = black_pixel;
  info->type   = windowtype;
  info->place  = place;
  info->name1  = "";
  info->name2  = "";
  info->mk_fn  = NULL;
  info->depvars = get_vardist(place, component, DEPENDENT);
  info->indepvars = get_vardist(place, component, INDEPENDENT);
  compdefs_info[compdefs_next++] = info;
}

/*PUBLIC*/ void diagram_draw_wires(w)
Window w;
{
  addx_next = 0;
  addy_next = 0;

  addx(x3   , yy10  , x5   , yy10  );
  addx(x6   , yy10  , x8   , yy10  );
  addx(x8   , y9    , x6   , y9    );
  addx(x5   , y9    , x3   , y9    );
/* Dumb--looking lines at corners instead: */
  addx(x13  , yn    , x13  , yy10-20);
  addx(x13+20, yy10 , x2   , yy10  );
  addx(x9   , yy10  , x14-20, yy10 );
  addx(x14  , yy10-20, x14  , yn   );
  addx(x14  , y6    , x14  , y9+20 );
  addx(x14-20, y9   , x9   , y9    );
  addx(x2   , y9    , x13+20, y9   );
  addx(x13  , y9+20 , x13  , y6    );
/*corners*/
  addx(x13  , y9+20 , x13+20, y9   );
  addx(x14-20, y9   , x14  , y9+20 );
  addx(x14  , yy10-20, x14-20, yy10 );
  addx(x13+20, yy10 , x13  , yy10-20);

  XDrawSegments(dpy, w, diagGC, alist, addx_next);
}

/*
 *
 * PUBLIC: mk_circuit_diagram()
 *
 */

Window mk_circuit_diagram(x, y)
int x,y;
{
  Window main_win;
  int n;
  XColor fore_color, back_color;
  XGCValues xgcv;
  unsigned long planemask;
  XSizeHints hints;
  XSetWindowAttributes attributes;

/*TAG
  printf("Entered mk_circuit_diagram in diagram.c\n");
*/
  x1 = 30; x10 = 600-30; /* outside edge of heart */
  x12 =120 ; x11 = 600-120; /* inside edge of heart */
  x2 = 120; x9 = 600-120; /* outside of outer boxes */
  x3 = 200; x8 = 600-200; /* inside of outer boxes */
  x4 = 230; x7 = 600-230; /* edges of center (system) box */
  x5 = 245; x6 = 600-245; /* edges of upper and lower middle boxes */

  yy1 = 30; y7 = 500-30; /* outer edges of upper and lower boxes */
  y2 = 150;y5 = 500-150;/* inner edges of upper and lower boxes */
  y3 = 210;y4 = 500-210;/* top and bottom of center (system) box */
  y6 = 150;y8 = 500-150; /* top and bottom of heart boxes */
  y9 = (y2-yy1)/2+yy1; /* upper connecting line */
  yy10 = (y7-y5)/2+y5;/* lower connecting line */

  dx = x12-x1; dy = y8-y6; /* dimensions of heart box */

  x13 = x1 + dx/2; /* middle of left heart box (connecting line)*/
  x14 = x10 - dx/2;/* middle of right heart box (connecting line)*/

  yn = y8-dy/5; /* location where connecting line touches heart */

  attributes.override_redirect = True;
  attributes.background_pixel = black_pixel;
  attributes.border_pixel = white_pixel;
  main_win = XCreateWindow(dpy, root, x, y, 600, 500, 5, CopyFromParent,
                           InputOutput, CopyFromParent, CWOverrideRedirect |
                           CWBorderPixel | CWBackPixel, &attributes);

  hints.x = x; hints.y = y; hints.width = 600; hints.height = 500;
```

```
hints.flags = USPosition | USSize; /* to bypass the window manager */
XSetNormalHints(dpy, main_win, &hints);
XSetTransientForHint(dpy, main_win, main_win);
/*TAG
printf("Created main_win 0x%x\n",main_win);
*/
/* Set the colors for the cursor. */
fore_color.red = 0;
fore_color.blue = 0;
fore_color.green = 0;
fore_color.flags = 0;

back_color.red = 0;
back_color.blue = 0;
back_color.green = 0;
back_color.flags = 0;

/* Get the cursor. */
left_ptr_cursor = XCreateFontCursor(dpy, XC_left_ptr);
/*TAG
printf("Created main_win cursor 0x%x\n",left_ptr_cursor);
*/
/* Change the cursor color. */
XRecolorCursor(dpy, left_ptr_cursor, &fore_color, &back_color);

/* main_win will use this cursor. */
XDefineCursor(dpy, main_win, left_ptr_cursor);

/* Load the font for the diagram. */
if (!diagram_font) diagram_font = XLoadQueryFont(dpy, DIAGRAMFONT);

/* Make the GCs for the windows. */
xgcv.foreground = white_pixel;
xgcv.background = black_pixel;
if (diagram_font)
  xgcv.font = diagram_font->fid;
xgcv.graphics_exposures = False;
mainGC = XCreateGC(dpy, main_win, GCForeground | GCBackground |
                   GCFont | GCGraphicsExposures, &xgcv);
xgcv.line_width = 2;
diagGC = XCreateGC(dpy, main_win, GCForeground | GCBackground | GCLineWidth,
                   &xgcv);

/*
 * XOR white and black to get the plane(s).
 */
planemask = white_pixel ^ black_pixel;

xgcv.plane_mask = planemask;
xgcv.function = GXinvert;
invertGC = XCreateGC(dpy, main_win, GCForeground | GCBackground |
                     GCFunction | GCPlaneMask, &xgcv);

if (!circuit_data) circuit_data = XUniqueContext();
if (!assoc_win)    assoc_win = XUniqueContext();
defs_next = icondefs_next = 0;
/* define icon associated windows (2 windows for each entry) */
box(main_win, x2, yy1, x3, y2, "Pulmonary", "Veins", mk_pv, PV);
box(main_win, x5, yy1, x6, y2, "Pulmonary", "Microcirculation", mk_cap, PC);
box(main_win, x8, yy1, x9, y2, "Pulmonary", "Arteries", mk_pa, PA);
win(main_win, x11+4, y6, x10+4, y8, "Right", "Ventricle", mk_rv, RV);
box(main_win, x8, y5, x9, y7, "Systemic", "Veins", mk_sv, SV);
box(main_win, x5, y5, x6, y7, "Systemic", "Microcirculation", mk_cap, SC);
box(main_win, x2, y5, x3, y7, "Systemic", "Arteries", mk_sa, SA);
win(main_win, x1, y6, x12, y8, "Left", "Ventricle", mk_lv, LV);
box(main_win, x4, y3, x7, y4, "System", "Parameters", mk_sys, SYSTEM);
box(main_win, 600-60, 500-40, 600-20, 500-20, "","Cancel", mk_exit, SYSTEM);

CreateWindowBatch(icondefs, icondefs_next);
CreateWindowBatch(defs, defs_next);
for (n=0; n<icondefs_next; n++) {
  XSaveContext(dpy, defs[n].self, assoc_win, (caddr_t)icondefs[n].self);
  defs[n].icon = icondefs[n].self;
  XSaveContext(dpy, icondefs[n].self, assoc_win, (caddr_t)defs[n].self);
  icondefs[n].icon = defs[n].self;
  XSaveContext(dpy, icondefs[n].self,circuit_data,(caddr_t)icondefs_info[n]);
}
for (n=0; n<defs_next; n++)
  XSaveContext(dpy, defs[n].self, circuit_data, (caddr_t)defs_info[n]);
defs_end = defs_next; icondefs_end = icondefs_next;

{
  button = defs[9].icon;
  /* Let modify parameters know which button is the exit button icon */
}

/* define 2nd level windows (sub-subwindows of main); these are InputOnly! */
compdefs_next = 0;
comp(defs[7].self,5, 54-16, 44, 54+16,  LV, VARCAPACITOR,VarCapacitorWindow);
comp(defs[7].self, 44-5,125 , 44+5, 140, LV, RESISTOR, ResistorWindow);
comp(defs[6].self, 40-10, 30, 40+10, 90, SA, CAPACITOR, CapacitorWindow);
comp(defs[5].self, 45, 59-10, 65, 59+10, SC, RESISTOR, ResistorWindow);
comp(defs[4].self, 20-10, 30, 20+10, 90, SV, CAPACITOR, CapacitorWindow);
comp(defs[4].self, 45, 59-10, 65, 59+10, SV, RESISTOR, ResistorWindow);
comp(defs[3].self,44, 109-16, 83, 109+16,RV,VARCAPACITOR,VarCapacitorWindow);
comp(defs[3].self, 44-5, 23, 44+5, 38,   RV, RESISTOR, ResistorWindow);
comp(defs[2].self, 40-10, 30, 40+10, 90, PA, CAPACITOR, CapacitorWindow);
comp(defs[1].self, 45, 59-10, 65, 59+10, PC, RESISTOR, ResistorWindow);
comp(defs[0].self, 40-10, 30, 40+10, 90, PV, CAPACITOR, CapacitorWindow);
comp(defs[0].self, 15, 59-10, 35, 59+10, PV, RESISTOR, ResistorWindow);

for (n=0; n<compdefs_next; n++) {
  compdefs[n].self = XCreateWindow(dpy, compdefs[n].parent,
                     compdefs[n].x, compdefs[n].y, compdefs[n].width,
                     compdefs[n].height, compdefs[n].borderwidth, 0, InputOnly,
                     CopyFromParent, 0, (XSetWindowAttributes *)0);
  XSaveContext(dpy, compdefs[n].self,circuit_data,(caddr_t)compdefs_info[n]);
}

if (!LHBdap || !RHBdap) {
/* Create pixmaps for left and right heart shapes */
#include "heart_l" /* the data from bitmap */
#include "heart_r"
    LHBdap = XCreatePixmapFromBitmapData(dpy, root, heart_l_bits,
                heart_l_width, heart_l_height, white_pixel, black_pixel,
                DefaultDepth(dpy,screen));

    RHBdap = XCreatePixmapFromBitmapData(dpy, root, heart_r_bits,
                heart_r_width, heart_r_height, white_pixel, black_pixel,
                DefaultDepth(dpy,screen));
  }
return(main_win);
```

```
}
```

modpar.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/X10.h>
#include <strings.h>
#include "../sim/CVDefs.h"
#include "../data/Parameter.h" /* for get_variable() */
#include "Xcv.h"

extern Display *dpy;
extern Window root;
extern int screen;
extern unsigned long white_pixel, black_pixel; /* Main.c */
extern double getval();

extern XFontStruct *diagram_font;
extern XContext circuit_data;
extern int advy_next;
extern Vertex vlist[];
extern Cursor left_ptr_cursor;
extern GC invertGC;
extern void init_icon(), diagram_draw_wires(), diagram_map_iconc();
extern XContext assoc_win;
extern Window init_circuit_diagram(); /* from diagram.c */
/*FORWARD*/ void choose_variable();

Window top_w;
GC labelGC;
static int done;
static unsigned long mask;
static GC imageGC = 0;
int nmanip_type; /* exported to diagram.c: init_sys() */
static int *place_ptr, *type_ptr;
/*FORWARD*/ static void getWinList();
static Window Diagram = 0;

modify_parameter(which, map_place, map_type)
int which; /* 0: modify independent vars, 1: plot dependent vars,2: plot both*/
int *map_place, *map_type;
{
  XEvent event;
  static winCount = 0;
  static Window winList[100] = {(Window) 0, };
  static Pixmap savedUnder = (Pixmap) 0;
  XGCValues xgcv;
  void handle_ExposeWindow(), handle_EnterWindow(),
       handle_LeaveWindow(), handle_ButtonPressed();
  XWindowAttributes att;
  int n;

  if (!Diagram) Diagram = init_circuit_diagram(100, 100);
  else XUnmapSubwindows(dpy, Diagram);
  xgcv.subwindow_mode = IncludeInferiors;
  if (!imageGC) imageGC = XCreateGC(dpy, root, GCSubwindowMode, &xgcv);
  if (!labelGC) {
    xgcv.foreground = whitepixel;
    xgcv.background = blackpixel;
    labelGC = XCreateGC(dpy, root, GCForeground | GCBackground, &xgcv);
  }
  nmanip_type = which;
  place_ptr = map_place;
  type_ptr = map_type;
  mask = ButtonPressMask | EnterWindowMask | LeaveWindowMask | ExposureMask;
  top_w = Diagram;
  XSelectInput(dpy, top_w, mask);
  /* POP-UP! This does a manual save-under, since I can't get the
   * automatic one to work.
   */
#ifndef NOGRABS
  XGrabServer(dpy); /* Don't let reality change while we've got it saved */
#endif NOGRABS
  XSync(dpy, False); /* Save the most up-to-date picture possible. */

  /* Calculate the footprint of this window. */
  XGetWindowAttributes(dpy, top_w, &att);
  if (!savedUnder) savedUnder =
    XCreatePixmap(dpy, root, att.width + 2*att.border_width,
                  att.height + 2*att.border_width, DefaultDepth(dpy,screen));
  XCopyArea(dpy, root, savedUnder, imageGC, att.x, att.y,
            att.width+2*att.border_width,att.height+2*att.border_width, 0, 0);
  XMapRaised(dpy, top_w);

  if (!winCount) getWinList(top_w, winList, &winCount);
  done = 0;
  while(!done) {
    for (;;) {
      XMaskEvent(dpy, mask, &event);
      for (n=0; n < winCount ; n++)
          if (((XEnterWindowEvent *)&event)->window == winList[n])
          goto comin;
    }
  comin:
    switch ((int) event.type) {
    case Expose:
      if (((XExposeEvent *)&event)->count == 0)
          /* Compress multiple rectangle exposure events, using only the last */
          handle_ExposeWindow((XExposeEvent *)&event);
      break;
    case EnterNotify:
      handle_EnterWindow((XEnterWindowEvent *)&event);
      break;
    case LeaveNotify:
      handle_LeaveWindow((XLeaveWindowEvent *)&event);
      break;
    case ButtonPress:
      handle_ButtonPressed((XButtonPressedEvent *)&event);
      break;
    default:
#ifdef DEBUG
      printf("Not handling event!  Type:  %d\n", event.type);
#endif
      break;
    }
  }
  /* POP DOWN! */
  XUnmapWindow(dpy, Diagram);
  /* Try to avoid redisplays, since we have saved all the bits. */
```

```
  XCopyArea(dpy, savedUnder, root, imageGC, 0, 0,
            xa.width + 2*xa.border_width, xa.height + 2*xa.border_width,
            xa.x, xa.y);
  XFreePixmap(dpy, savedUnder); savedUnder = (Pixmap) 0;
  {/* Discard all accumulated exposure events. */
   XEvent ev;
   XSync(dpy, False);
   while (XCheckTypedEvent(dpy, Expose, &ev) == True)
     ; /* discard them since we repainted the screen instead. */
  }
#ifndef NOGRABS
  XUngrabServer(dpy);
#endif NOGRABS
  return;
}

/* This is a recursive, pre-order traversal of all windows with win as
 * ancestor.  The routine builds a list in winList and keeps a count in
 * countPtr, both returned values.  Space for winList & countPtr must be
 * allocated by calling fn.
 */
static void getWinList(win, winList, countPtr)
Window win;
Window *winList; /*RETURN*/
int *countPtr;   /*RETURN*/
{
  unsigned int nchildren;
  Window *children, parent, root_return;
  XQueryTree(dpy, win, &root_return, &parent, &children, &nchildren);
  winList[(*countPtr)++] = win;
  XSaveContext(dpy, win, xtab);
  while (nchildren--)
    getWinList(children[nchildren], winList, countPtr);
  XFree((char *)children);
}

static void handle_ExposeWindow(e)
XExposeEvent *e;
{
  CircInfo *cinfo;
  Window w;
  char *label;

  w = e->window;
  if (w == top_w) {
    diagram_map_icons();
    diagram_draw_wires(top_w);
    if (param_type == 0) label = "Choose Parameter to Modify";
    else                 label = "Choose Variable to Plot";
    diagram_draw_label(top_w, label);
  } else {
    if (!XFindContext(dpy, w, circuit_data, (caddr_t *)&cinfo)) {
      if (cinfo && cinfo->exk_fn)
          (cinfo->exk_fn)(w);
      XMapSubwindows(dpy, w);
    }
  }
}

Window button = 0;
static void handle_ButtonPressed(e)
XButtonEvent *e;
{
  Window w, assoc;
  CircInfo *cinfo;

  w = e->window;

  if (w == top_w) {
    goof();
  } else if (w == button) {
    *place_ptr = *type_ptr = -1; /* Fill return values with null choice */
    done = 1;
  } else {

    if (!XFindContext(dpy, w, circuit_data, (caddr_t *)&cinfo)) {
      if (cinfo) switch(cinfo->type) {
      case IconWindow:
          if (!XFindContext(dpy, w, assoc_win, (caddr_t *)&assoc)) {
            XUnmapWindow(dpy, w);
            XMapRaised(dpy, assoc);
          }
          break;
      case CircuitWindow:
      case DiodeWindow:
          goof(); /*assup...ar error foup*/
          break;
      default:
          if (cinfo)
            choose_variable(cinfo, param_type);
          break;
      }
    }
  }
}

static void handle_EnterWindow(e)
XCrossingEvent *e;
{
  Window w = e->window;
  if (w == top_w) {
    ; /*assup*/
  } else switch (e->detail) {
    case NotifyAncestor:
    case NotifyNonlinear:
    case NotifyNonlinearVirtual:
      hilite(w);
      break;
    case NotifyInferior:
      /*assup*/
      break;
    }
}

static void handle_LeaveWindow(e)
XCrossingEvent *e;
{
  Window w, icon;
  CircInfo *cinfo;
  XWindowAttributes win_attr;
  w = e->window;
  if (w == top_w) {
```

```
    : /* do nothing */
  } else {
   if (XFindContext(dpy, w, circuit_data, (caddr_t *)&cinfo)) {
    if (cinfo && cinfo->type == IconWindow) {
        XGetWindowAttributes(dpy, w, &win_attr);
        if (win_attr.map_state == IsUnmapped) {
/*TAG ... I think we would rather do nothing here.
         if (XFindContext(dpy, w, assoc_win, &icon)) {
          XUnmapWindow(dpy, icon);
          XMapRaised(dpy, w);
         }
*/ ;
        } else {
          do_hilite(w);
        }
    } else switch (e->detail) {
    case NotifyNonlinear:
    case NotifyNonlinearVirtual:
    case NotifyAncestor:
        if (cinfo && cinfo->type == CircuitWindow) {
         if (XFindContext(dpy, w, assoc_win, (caddr_t *)&icon)) {
          XUnmapWindow(dpy, w);
          XMapRaised(dpy, icon);
         }
        } else {
         do_hilite(w);
        }
        break;
    case NotifyInferior:
        /*map*/
        break;
   }
  }
 }
}

static hilite(w)
Window w;
{
 CircInfo *cinfo;

 if (XFindContext(dpy, w, circuit_data, (caddr_t *)&cinfo)) {
  if (cinfo) switch(cinfo->type) {
  case LowWindow:
   if (cinfo->unk_fn) (cinfo->unk_fn)(w);
   XMapSubwindows(dpy, w);
   XFillRectangle(dpy, w, invertGC, 0, 0, cinfo->width,
                  cinfo->height);
/*TAG
   printf("Highlight this window! w = 0x%lx\n", w);
*/
   break;
  case ResistorWindow:
  case CapacitorWindow:
  case VarCapacitorWindow:
   /* These are InputOnly windows, so must instead write to parent */
   { Window root_ret, parent, *children;
     unsigned int nchildren;
     int x, y;
     XQueryTree(dpy, w, &root_ret, &parent, &children, &nchildren);
     XFree((char *)children);
     XTranslateCoordinates(dpy, w, parent, 0, 0, &x, &y, &root_ret);
     XFillRectangle(dpy, parent,invertGC, x, y, cinfo->width,cinfo->height);
     break;
   }
  case CircuitWindow:
  case DiodeWindow:
   /* not hilited */
   break;
  }
 }
}

static do_hilite(w)
Window w;
{
 CircInfo *cinfo;
 hilite(w);
 if (XFindContext(dpy, w, circuit_data, (caddr_t *)&cinfo)) {
  if (cinfo->unk_fn) (cinfo->unk_fn)(w);
  XMapSubwindows(dpy, w);
 }
}

static good()/* an error button press etc. */
{
/*
printf("GOOD!\n");
*/
}

void choose_variable(cinfo, mode) /* used locally and by diagram.c:mkyyu()*/
CircInfo *cinfo;
int mode;
{
 VarData *vd;

 Window w, w2, button[10], button_done, root_return, child_return;
 char *str;
 int i, n, var[20];
 XEvent xevent;
 XButtonEvent *bevent = (XButtonEvent *) &xevent;
 int done2, x, y, root_x, root_y;
 unsigned long mask;
 unsigned int mask_return;
 static GC textGC = 0;
 XGCValues xgcv;
 static Pixmap saved_Under = (Pixmap) 0;
 XWindowAttributes att;
 XSizeHints hints;
 XSetWindowAttributes attributes;
 XQueryPointer(dpy, root, &root_return, &child_return, &root_x,
               &root_y, &x, &y, &mask_return);
 attributes.override_redirect = True;
 attributes.background_pixel = white_pixel;
 attributes.border_pixel = black_pixel;
 w = XCreateWindow(dpy, root, x, y, 300, 120, 2, CopyFromParent, InputOutput,
                   CopyFromParent, CWOverrideRedirect | CWBackPixel |
                   CWBorderPixel, &attributes);
 hints.x = x; hints.y = y; hints.width = 300; hints.height = 120;
 hints.flags = USPosition | USSize;
 XSetNormalHints(dpy, w, &hints);
```

```
XSetTransientForHint(dpy, w, Diagram);
XDefineCursor(dpy, w, left_ptr_cursor);
n=0;
if (mode != 0) for (n = 0; cinfo->depvars[n] != ENDVAR; n++)
  vars[n] = cinfo->depvars[n];
if (mode == 2) vars[n++] = 999;
if (mode != 1) for (i = n; cinfo->indepvars[n-i] != ENDVAR; n++)
  vars[n] = cinfo->indepvars[n-i];
vars[n] = ENDVAR;

if (mode == 0) str = "Modification Choices:";
else           str = "Plotting Choices:";
n = 0;
while (vars[n] != ENDVAR) {
  if (vars[n] == 999) button[n] = (Window)0; /*space holder*/
  else button[n] = XCreateSimpleWindow(dpy,w, 10, 30+15*n, 10, 10, 1,
                                       white_pixel, black_pixel);
  n++;
}
button_done = XCreateSimpleWindow(dpy,w, 10, 30+15*n++, 10, 10, 1,
                                  white_pixel, black_pixel);
XRaiseWindow(dpy, w, 300, 30+15*n);

mask = ButtonPressMask | EnterWindowMask | LeaveWindowMask | ExposureMask;
XSelectInput(dpy, w, mask);
XMapSubwindows(dpy, w);
{
  /* POP--UP! This does a manual save--under, since I can't get the
   * automatic one to work.
   */
  XSync(dpy, False); /* Save the most up--to--date picture possible. */
  /* Calculate the footprint of this window. */
  XGetWindowAttributes(dpy, w, &att);
  savedUnder =
    XCreatePixmap(dpy, root, att.width + 2*att.border_width,
                  att.height + 2*att.border_width, DefaultDepth(dpy,screen));
  XCopyArea(dpy, root, savedUnder, imageGC, att.x, att.y,
            att.width+2*att.border_width,att.height+2*att.border_width,0,0);
}
XMapRaised(dpy, w);
done2 = 0;

xgcv.font = diagram_font->fid;
xgcv.foreground = black_pixel;
xgcv.background = white_pixel;
if (!textGC) textGC = XCreateGC(dpy, root,
                                GCFont | GCForeground | GCBackground, &xgcv);
while(!done2) {
  XWindowEvent(dpy, w, mask, &xevent);
  switch (&xit) xevent.type) {
  case Expose:
    if ((XExposeEvent *)&xevent)->count == 0) {
        /* Compress multiple rectangle exposure events, using only the last */
        XDrawString(dpy, w, textGC, 10, 10, str, strlen(str));
        for (n=0; vars[n] != ENDVAR; n++) {
          if (vars[n] == 999) XDrawString(dpy, w, textGC, 30, 40+15*n,
            "-------------------------------------------------", 49);
          else {
            vd = get_variate(cinfo->plane, vars[n]);
            XDrawString(dpy, w, textGC, 30, 40+15*n, vd->name,
                        strlen(vd->name));
          }
        }
    }
    break;
  case EnterNotify:
    break;
  case LeaveNotify:
    break;
  case ButtonPress:
    w2 = bevent->window;
    if (bevent->subwindow) w2 = bevent->subwindow;
    if (w2 == w) break;
    if (w2 == button_done) {done2 = 1; break;}
    for (n=0; n<10; n++) if (button[n] == w2) break;
  /* If mode == 0, should make a NumberLine instead! */
    /*return to set up plot with this variable */
    *plane_ptr = cinfo->plane;
    *type_ptr = vars[n];
    done2 = 1;
    done = 1;
    break;
  }
}
/* POP DOWN! */
XDestroyWindow(dpy, w);
/* Try to avoid redisplays, since we have saved all the bits. */
XCopyArea(dpy, savedUnder, root, imageGC, 0, 0,
          att.width + 2*att.border_width, att.height + 2*att.border_width,
          att.x, att.y);
XFreePixmap(dpy, savedUnder); savedUnder = (Pixmap) 0;
/* calling routine must discard external exposure events. */
return;
}


diagram_draw_label(w, label)
Window w;
char *label;
{
  static XFontStruct *largeFont = NULL;
  Window root_return;
  int x_return, y_return;
  unsigned int width_return, height_return,
               border_return, depth_return;
  int string_x;
  XGCValues xgcv;

  XGetGeometry(dpy, w, &root_return, &x_return, &y_return,
               &width_return, &height_return,
               &border_return, &depth_return);
  if (!largeFont) largeFont = XLoadQueryFont(dpy, LARGEFONT);
  if (largeFont) {
    xgcv.font = largeFont->fid;
    XChangeGC(dpy, labelGC, GCFont, &xgcv);
    string_x = width_return/2 - XTextWidth(largeFont, label, strlen(label))/2;
  } else
    string_x = width_return/2 - 10;
  XDrawString(dpy, w, labelGC, string_x, 30, label, strlen(label));
}
```

test.mp.c

```
#include <X11/Xlib.h>
#include <stdio.h>
#include "JointCVDefs.h"

extern Window mk_circuit_diagram();
extern SIMULATION *getsimulation();

Display *dpy;
int screen;
unsigned long white_pixel, black_pixel;                    2200
Window root;

main()
{
  SIMULATION *parameters;
  int place, type;

  if ((dpy = XOpenDisplay(NULL)) == NULL)
    printf("Can't open display!\n");
  screen = DefaultScreen(dpy);                             2210
  root = RootWindow(dpy, screen);
  white_pixel = WhitePixel(dpy, screen);
  black_pixel = BlackPixel(dpy, screen);
  XSynchronize(dpy, 1);
  LoadParameterData("/mit/hst/dev/data/db");
  parameters = getsimulation(NORMAL);
  initeqvars(parameters);
  beamt();
  modify_parameter(0, &place, &type);
  modify_parameter(1, &place, &type);                      2220
  printf ("Place %d,  type %d\n",place, type);
  printf("Done!\n");
}
```

C.8 Main event loop

These files describe the main event loop for the program, which consists of simulating several time-steps, sending data to the plot routines, and receiving and processing all X events. In addition, housekeeping functions such as starting, stopping, and resetting the mathematical simulation are located here.

Event.h

```
/*
 * For data from pipe
 *
 */

#define DATA 1
#define PARAMETER 2

/*
 * Public Procedures                                       10
 */
extern void ResetSimulator(/* SIMULATION *parameters; */);
extern int ResumeSimulator();
extern void StopSimulator();
extern void SendToSim(/* int place, type; double value; */);
extern void MakeSimulator(/* SIMULATION *parameters;double startime; */);
extern void SimFlush();
extern void SimDataRing(/* double *pressures */);
extern void UpdateParamWindow();
extern int Stopped;                                       20
extern void PostErrorString(/* char *s */);
extern void PostErrorFile(/* char *s */);
extern int fpe_in_simulate(); /* Error handler for floating-point errors. */

extern void AddDataWidget(); /* widget, modProc, dataProc */
/*
 * Widget widget;
 * void (*modProc)();
 * void (*dataProc)();
 */                                                       30

extern void DeleteDataWidget(); /* widget */
/*
 * Widget widget;
 */

/*
 * Processes X events and also checks the data queue and updates data plots.
 * This is the main execution loop.
 */                                                       40

extern void DispatchDataAndEvents();
```

Event.c

```
#ifndef lint
static char *rcsid="$Header: /mit/hst/src/event/RCS/Event.c,v 2.4 91/01/10 17:18:12 tldavis Exp $";
#endif
/*
 * Event.c: Copyright 1990 Timothy L. Davis
 *
 * These functions deal with interaction between the simulation,
 * the user interface, and the operating system.           50
 * The main top-level loop DispatchDataAndEvents() is also here.
 *
 * To use saphyr windowgrams to report errors, define the symbol "ZEPHYR".
 * To use a pipe to store simulation data, define the symbol "USEPIPE".
 *
 * $Log:            Event.c,v $
 * Revision 2.4  91/01/10  17:18:12  tldavis
 * Fixed problem with POLLING.  If there is nothing to do,
 * the simulator will now BLOCK in XtNextEvent() instead of constantly
 * checking XPending() and checking if more simulation data is needed.   60
 * (This bug was in DispatchDataAndEvents().)
 * .
 *
 * Revision 2.3  90/08/28  14:36:15  tldavis
 * Fixed error file posting.
 *
 * Revision 2.1  90/08/18  18:32:20  tldavis
 * General cleanup.
 *
 */                                                       70

#include <X11/Intrinsic.h>
#include <fcntl.h>
```

```
#include <stdio.h>

#include "../sim/CVDefs.h"
#include "../data/Parameter.h"
#include "Event.h"

extern char *CVLib; /* from Main.c */

/* xemm includes */
#include <sys/types.h>
#include <sys/time.h>
/* end xemm includes */

#ifdef NOTDEF
#define ZEPHYR /* Use zwrite for Project Athena error notification. */
#endif NOTDEF
extern int simulate();
extern Display *dpy;
extern Widget paramWidget;    /* from ../cv/Interface.c */
extern caddr_t paramWinData; /* from ../cv/Interface.c */
extern void ShowParameters();/* from ../cv/Interface.c */
extern char *malloc(), realloc();

/*
 * record of window and update/data procs
 */
typedef struct {
  Widget widget;
  void (*modProc)();
  void (*dataProc)();
} DataWidgetRec;


static int FPERROR = 0;
/*PUBLIC*/ /*ARGSUSED*/
int fpe_in_simulate()
{
  PostHaveString("Floating Point Error in simulate.c");
  FPERROR = 1; /* Not an extern variable, so OK for Sun--4 -O3 optimization */
  return(0);
}

/*PUBLIC*/
void PostHaveString(s)
    char *s;
{
#ifdef ZEPHYR
  char cc[256];
  sprintf(cc, "zwrite -q $USER -m %s\n",s);
  system(cc);
#else /* No zephyr windowgrams */
  XtWarning(s);
#endif ZEPHYR
}

/*PUBLIC*/
void PostHaveFile(s)
    char *s;
{
#ifdef ZEPHYR
  char cc[256];
  sprintf(cc,"zwrite -q $USER < %s",s);
  system(cc);
#else /* No zephyr windowgrams, use stderr */
  int c;
  FILE *file;
  char buf[2048], *p, *end;

  XtWarning("SIMULATOR ERROR:\n");
  if ((file = fopen(s,"r")) >0) {
    p = buf;
    end = &buf[2047];
    while ((c=getc(file))!=EOF && p < end) *p++ = (char) c;
    fclose(file);
    *p++ = '\n';
    XtWarning(buf);
  }
#endif ZEPHYR
}

static double x[16];
/*PUBLIC*/ int Stopped;

static DataWidgetRec *DWlist = (DataWidgetRec *)0, *dwLast;
static int DWlen = 0, DWactive = 0;

/*
 * Register a widget to receive simulation data from DispatchData()
 * Takes widget window, parameter modification handler modProc(), and
 * data reception handler dataProc():
 *
 * void (*modProc)(Widget widget,double x[], int  place,int type, double value);
 * void (*dataProc)(Widget widget, double[] parameters);
 */
/*PUBLIC*/ void AddDataWidget(widget, modProc, dataProc)
Widget widget;
void (*modProc)(), (*dataProc)();
{
  int i = DWlen;
  DWactive++;
  if (!i) {
    while (i--)
      if (DWlist[i].widget == (Widget) 0) {/*0 means unused record (deleted) */
          DWlist[i].widget = widget;
          DWlist[i].modProc = modProc;
          DWlist[i].dataProc = dataProc;
          return;
      }
    DWlist = (DataWidgetRec *)realloc((char *)DWlist,
                                      (2+DWlen)*sizeof(DataWidgetRec));
  } else {
    DWlist = (DataWidgetRec *)malloc((2+DWlen)*sizeof(DataWidgetRec));
  }
  DWlist[DWlen].widget = widget;
  DWlist[DWlen].modProc = modProc;
  DWlist[DWlen].dataProc = dataProc;
  DWlist[++DWlen].widget = (Widget)0; /* end of list */
  dwLast = &DWlist[DWlen];
}

/*
 * Remove widget from the list of those receiving data packets
```

```
 */
/*PUBLIC*/ void DeleteDataWidget(widget)
Widget widget;
{
 int i = DWlen;
 while (i--)
  if (DWlist[i].widget &&
        (DWlist[i].widget == widget || XtParent(DWlist[i].widget) == widget))
   {
    DWlist[i].widget = (Widget)0; /* delete this entry */
    DWactive--;
   }
}

void UpdateParamWindow()
{
 if (paramWidget) ShowParameters((Widget)0, paramWinData, (caddr_t)0);
}

/*
 * Get data from pipe fd, and broadcast it to all widgets which have requested
 * a simulation data stream (used in DispatchDataAndEvents).
 */
static double currentTime = 0.0; /* the current time for Reset() */

#ifdef USEPIPE
static void DispatchData(fd, dataWidget)
int fd;
DataWidgetRec *dataWidget;
{
 char dataType;
 double pressures[10];
 fd_set readfds;
 struct timeval timeout;
 DataWidgetRec *dw;

 if (DWactive) return;
 if (DWactive < 0 || DWlen < 0) fprintf(stderr,"Bad DataPlot list!\n");
 FD_ZERO(&readfds);
 FD_SET(fd, &readfds);
 timerclear(&timeout);
 while (select(FD_SETSIZE, &readfds,(fd_set *)0,(fd_set *)0,&timeout) != 0) {
  read(fd, &dataType, sizeof(dataType));
  switch (dataType) {
  case DATA:
   if (read(fd,pressures,sizeof(pressures[0])*10)==sizeof(pressures[0])*10){
        currentTime = pressures[6];
        for (dw = dataWidget; dw != dwLast; dw++)
         if (dw->widget) (*dw->dataProc)(dw->widget, pressures);
   }
   break;
  }
 }
}
#else not USEPIPE

#define SIMBUFLENGTH 110
static double SimBuffer[SIMBUFLENGTH]; /* Storage for simulation data between
                                        * data production and plotting.
                                        */
static int SimBufHead = 0, SimBufTail = 0;

static void DispatchData(dataWidget)
DataWidgetRec *dataWidget;
{
 double pressures[10];
 DataWidgetRec *dw;
 int i;

 if (DWactive) return;
 if (DWactive < 0 || DWlen < 0) fprintf(stderr,"Bad DataPlot list!\n");
 while (SimBufHead != SimBufTail) {
  for (i=0; i<10; i++) {
   pressures[i] = SimBuffer[SimBufHead++];
   if (SimBufHead >= SIMBUFLENGTH) SimBufHead -= SIMBUFLENGTH;
   if (SimBufHead == SimBufTail)
       i=100;
  }
  if (i < 100) { /* Got a complete set of pressures to dispatch */
   currentTime = pressures[6];
   for (dw = dataWidget; dw != dwLast; dw++)
       if (dw->widget) (*dw->dataProc)(dw->widget, pressures);
  }
 }
}
#endif USEPIPE

static Boolean Flushing=False; /* Whether to flush simulation data or use it*/

/*PUBLIC*/ void SimDataSink(pressures)
double *pressures;
{
 if (Flushing==True || PPERROR) return;
 else
#ifndef USEPIPE
  {
   register int i;
   for (i = 0; i < 10 ; i++) {
       SimBuffer[SimBufTail++] = pressures[i];
       if (SimBufTail >= SIMBUFLENGTH) SimBufTail -= SIMBUFLENGTH;
       if (SimBufTail == SimBufHead) {
        PostErrorString("Simulation Buffer Overflow!");
        SimBufTail--;
       }
   }
  }
#else USEPIPE
  {
   int c, len;
   char *start;
   static char header=DATA;
   while (write(SimToPlotPipe[1], &header, sizeof(char)) < sizeof(char)) ;
   start = (char *) pressures;
   len=sizeof(pressures[0])*10;
   while ((c=write(SimToPlotPipe[1], start, len)) != len)
       { len -= c; start += c; }
  }
#endif USEPIPE
}

/*****************************************************************/
/*
```

```
 *
 * The following help the simulator to communicate with the plot process.
 *
 */

#ifdef USEPIPE
static int SimToPlotPipe[2]; /* pipe from simulator -> plotting process */
#endif
/*
 * This is the main top-level control loop.  Process all X events, then
 * get the next data from the simulator process if there are any active
 * plots.  If no active plots exist, the loop will block in XtNextEvent()
 * processing only user interactions, without polling, to save CPU time
 * for other possible processes.
 */
/*PUBLIC*/ void DispatchDataAndEvents() /* never returns */
{
  XEvent ev;
  double old_z[10];
  int failures = 0, i;

#ifdef USEPIPE
  if (fd) SimToPlotPipe[0] = fd;
#endif
  for (;;) {
    while (XPending(dpy) || !DWactive || Stopped)
      /* Allow to block only if we have no reason to calculate data */
      { XNextEvent(dpy, &ev); XtDispatchEvent(&ev); }

#ifdef USEPIPE
    DispatchData(SimToPlotPipe[0], DWlist);
#else
    DispatchData(DWlist);
#endif
    /* Compute the next bit of simulation data, taking care to
     * restart on the event of a simulation failure.
     */
    z[6] = currentTime;
    Plunking = False;
    if (simulate(z, 10)) {
      for (i = 0; i < 10; i++) old_z[i] = z[i];
      failures = 0;
      currentTime = z[6];
    } else if (++failures > 50 || PFERROR) {
      /* hope the user resets a parameter, but if not */
      char path[1024];
      sprintf(path,"%s/convergence.failure", CVLib);
      PostScriptFile(path);
      Stopped = 1; /* quit simulation process; user must re-start */
      failures = 0;
      PFERROR = 0;
      for (i = 0; i<10; i++) z[i] = old_z[i];
    }
  }
}

/*PUBLIC*/ void SimFlush() /* Clears data queue. */
{
#ifdef USEPIPE
  DispatchData(SimToPlotPipe[0], DWlist);
#else
  DispatchData(DWlist);
#endif
}

/*
 * PUBLIC routines handling communication with the SIMULATOR through
 * Unix pipes or a manually-handled buffer.
 *
 */
/*PUBLIC*/ void SendToSim(place, type, value) /*send param change to sim.proc*/
int place, type;
double value;
{
  setval(z, place, type, value);
}

/*PUBLIC*/ void MakeSimulator(parameters, startime)
SIMULATION *parameters;
double startime;
{
  static int i;
#ifdef USEPIPE
  static int devnull = -1;
  if (pipe(SimToPlotPipe) != 0) {
    fprintf(stderr, "Pipe creation failed.\n");
  }
  if (devnull < 0) devnull = open("/dev/null",O_WRONLY, 0);
#endif USEPIPE
  initsqrvars(parameters);
  setupCoeffs();
  bcount();
  setinitz(parameters, z); /* sets z[9] to 0, start a new host. */
  Plunking = True;
  for (i=0; i < 4; i++)
    simulate(z, TBLSIZE);
  Plunking = False;
  Stopped = 0;
  z[6] = currentTime = startime;
}

/* send signal to pause simulation */
/*PUBLIC*/ void StopSimulator()
{
  Stopped = 1;
}

/* resume simulation where we left off, no steady-state finding. */
/*PUBLIC*/ int ResumeSimulator()
{
  Stopped = 0;
#ifdef USEPIPE
  return(SimToPlotPipe[0]);
#else
  return(1);
#endif
}

/* Reset simulator with a new data state.  Find new steady state. */
/*PUBLIC*/
void ResetSimulator(parameters)
SIMULATION *parameters;
```

```
{
#ifdef USEPIPE
  close(SimToPlotPipe[0]); /* discard the data in the pipe. */
  close(SimToPlotPipe[1]);
#endif
  MakeSimulate(parameters, currentTime); /* Continue from current time */
}
```

C.9 Run-time library

These files must be on the system in a directory pointed to by the CVLIB environment variable (presently `/mit/hst/cvLib`). `parameter.db` defines the parameter and variable names, limits for plotting and number-lines, and locations on the circuit diagram. `XResources` contains default sizes, locations, and colors of the X window system interface windows. The files `hist.*` are the individual patient histories which show up as as the case number of the digit in their file name. They should be encrypted using the `mkpat` script and placed in a subdirectory `Patients` .

parameter.db

```
# Database for variable names, default values, allowed minimum, maximum, and graphing default min and max.
#
# These are simulation parameters (independent variables)
#
#       :Place  :Device  :Type    :Dep/Ind:Default:Min     :Max    :Gmin   :Gmax   :Name
#
        :l      :vcap    :cvys    :i       :0.4    :0.2    :20.0   :0.0    :2.0    :LV Min. Systolic Capacitance
        :r      :vcap    :cvys    :i       :1.2    :0.2    :49.0   :0.0    :4.0    :RV Min. Systolic Capacitance

        :l      :vcap    :cdias   :i       :10.0   :0.2    :20.0   :0.0    :20.0   :LV Max. Diastolic Capacitance
        :r      :vcap    :cdias   :i       :29.0   :0.2    :49.0   :0.0    :40.0   :RV Max. Diastolic Capacitance

        :a      :cap     :cap     :i       :1.6    :0.1    :20.0   :0.0    :6.0    :Arterial Capacitance
        :pa     :cap     :cap     :i       :4.3    :0.1    :20.0   :0.0    :6.0    :Pulmonary Artery Capacitance
        :v      :cap     :cap     :i       :108.0  :0.1    :500.0  :0.0    :200.0  :Venous Capacitance
        :pv     :cap     :cap     :i       :8.4    :0.1    :49.0   :0.0    :20.0   :Pulmonary Venous Capacitance

        :l      :vcap    :zvol    :i       :15.0   :0.0    :1000.0 :0.0    :30.0   :LV Zero-Pressure Volume
        :r      :vcap    :zvol    :i       :15.0   :0.0    :1000.0 :0.0    :33.0   :RV Zero-Pressure Volume
        :a      :cap     :zvol    :i       :715.0  :0.0    :5000.0 :0.0    :1500.0 :Arterial Zero-Pressure Volume
        :pa     :cap     :zvol    :i       :98.0   :0.0    :5000.0 :0.0    :200.0  :Pulm. Art. Zero-Pressure Volume
        :v      :cap     :zvol    :i       :2598.0 :0.0    :10000.0:0.0    :5000.0 :Venous Zero-Pressure Volume
        :pv     :cap     :zvol    :i       :490.0  :0.0    :5000.0 :0.0    :1000.0 :Pulm. Ven. Zero-Pressure Volume

        :l      :res     :res     :i       :0.006  :0.004  :2.0    :0.0    :0.01   :LV Outflow Resistance
        :r      :res     :res     :i       :0.005  :0.0001 :2.0    :0.0    :0.01   :RV Outflow Resistance
        :micro  :res     :res     :i       :1.0    :0.001  :10.0   :0.0    :5.0    :Arteriolar Resistance
        :pmicro :res     :res     :i       :0.08   :0.0001 :2.0    :0.0    :0.1    :Pulmonary Vascular Resistance
        :v      :res     :res     :i       :0.05   :0.0001 :2.0    :0.0    :0.1    :Venous Resistance
        :pv     :res     :res     :i       :0.010  :0.0005 :2.0    :0.0    :0.1    :Left Heart Inflow Resistance

        .sys    :system  :hr      :i       :72     :10.0   :250.0  :30.0   :150.0  :Heart Rate
        :sys    :system  :volume  :i       :5000.0 :100.0  :10000.0:0.0    :10000.0:Total Blood Volume
        :sys    :system  :pressure         :-4.0   :-50.0  :50.0   :-10.0  :10.0   :Pulmonary Tissue-Thoracic Pressure

# These are all Dependent (computed) variables
#
#       :Place  :Device  :Type    :Dep/Ind:Default:Min     :Max    :Gmin   :Gmax   :Name
#
        :l      :vcap    :pressure         :0      :-100   :500    :0.0    :150.0  :LV Pressure
        :r      :vcap    :pressure         :0      :-100   :500    :0.0    :150.0  :RV Pressure
        :a      :cap     :pressure         :0      :-100   :500    :0.0    :150.0  :Systemic Arterial Pressure
        :pa     :cap     :pressure         :0      :-100   :500    :-10.0  :30.0   :Pulmonary Arterial Pressure
        :v      :cap     :pressure         :0      :-100   :500    :0.0    :20.0   :Systemic Venous Pressure
        :pv     :cap     :pressure         :0      :-100   :500    :0.0    :20.0   :Pulmonary Venous Pressure

        :l      :vcap    :volume  :d       :0      :0      :5000   :0.0    :200.0  :LV Volume
        :r      :vcap    :volume  :d       :0      :0      :5000   :0.0    :200.0  :RV Volume
        :a      :cap     :volume  :d       :0      :0      :5000   :700.0  :1000.0 :Systemic Arterial Volume
        :pa     :cap     :volume  :d       :0      :0      :5000   :100.0  :370.0  :Pulmonary Arterial Volume
        :v      :cap     :volume  :d       :0      :0      :5000   :3100.0 :3250.0 :Systemic Venous Volume
        :pv     :cap     :volume  :d       :0      :0      :5000   :500.0  :750.0  :Pulmonary Venous Volume

        :l      :res     :flow    :d       :0      :0      :10000  :-100.0 :1000   :LV Outflow
        :r      :res     :flow    :d       :0      :0      :10000  :-100.0 :2000   :RV Outflow
        :micro  :res     :flow    :d       :0      :0      :10000  :0.0    :200    :Systemic Microcirculatory Flow
        :pmicro :res     :flow    :d       :0      :0      :10000  :0.0    :200    :Pulmonary Microcirculatory Flow
        :v      :res     :flow    :d       :0      :0      :10000  :-100.0 :200    :RV Inflow
        :pv     :res     :flow    :d       :0      :0      :10000  :-100.0 :1000   :LV Inflow

        :l      :vcap    :cap     :d       :0      :0      :10000  :0.0    :10.0   :LV Instantaneous Capacitance
        :r      :vcap    :cap     :d       :0      :0      :10000  :0.0    :20.0   :RV Instantaneous Capacitance
        :v      :res     :cdias   :d       :0      :-100   :500    :-10    :20.0   :Right Atrial Pressure
        :pv     :res     :cdias   :d       :0      :-100   :500    :0.0    :20.0   :Left Atrial Pressure

        :sys    :system  :time    :d       :0.0    :0.0    :9999999:0.0    :5.0    :Time
```

XResources

```
#ifdef COLOR
cvs*Plot.color1: red
cvs*Plot.color2: forest green
cvs*Plot.color3: blue
cvs*Plot.plotback: white
cvs*Plot.background: light blue
cvs*Plot.foreground: black
cvs*Plot.Command.foreground: salmon
cvs*Plot.Command.background: tan
cvs*Dialog.Command.foreground: black
cvs*Dialog.Command.background: red
cvs*History*foreground: navy
cvs*CommandButton.foreground: salmon
cvs*CommandButton.background: tan
cvs*Box.background: sky blue
cvs*Form.background: sky blue
cvs*Form.Label.background: sky blue
cvs*plotForm*DoPlot.foreground: salmon
cvs*plotForm*DoPlot.background: tan
cvs*plotForm*Cancel.foreground: salmon
cvs*plotForm*Cancel.background: tan
cvs*foreground: brown
cvs*background: wheat
cvs*NumberLine*foreground: navy
cvs*NumberLine*Label.foreground: salmon
cvs*NumberLine.background: white
cvs*NumberLine*Label.background: white
cvs*NumberLine*Command.foreground: salmon
cvs*NumberLine*Command.background: tan
cvs*NumberLine*Text*foreground: navy
```

```
cvs*NumberLine*Text*background: wheat

#else /* no color (b/w), so make sure things are readable. */
cvs*Plot.color1: black
cvs*Plot.color2: black
cvs*Plot.color3: black
cvs*Plot.plotBack: white
cvs*Plot.foreground: black
cvs*Plot.background: white
cvs*History*foreground: white
cvs*History*background: black
cvs*NumberLine.foreground: black
cvs*NumberLine.background: white
cvs*NumberLine*Command.foreground: white
cvs*NumberLine*Command.background: black
cvs*NumberLine*Text*foreground: black
cvs*NumberLine*Text*background: white
cvs*NumberLine*Label.foreground: black
cvs*NumberLine*Label.background: white
#endif

cvs*cursor: heart
cvs*Text*cursor: xterm
cvs*Command.cursor: lefthuman
cvs*Dialog.Quit.cursor: pirate
cvs*Dialog.Continue.cursor: gumby

cvs*commandButton.shapeStyle: Oval
cvs*Plot.Command.shapeStyle: RoundedRectangle
cvs*NumberLine.Command.shapeStyle: RoundedRectangle
cvs*plotForm.Form.Cancel.shapeStyle: Oval
cvs*plotForm.Form.DoPlot.shapeStyle: Oval
cvs*plotForm*shapeStyle: Oval
cvs*Dialog.Command.shapeStyle: Oval

#if (RELEASE == 4)
cvs*commandButton.font:    -*-helvetica-medium-r-normal--*-*-100-100-100-*-*-*-*-*
cvs*Dialog.Command.font:   -*-helvetica-medium-o-normal--*-*-240-100-100-*-*-*-*-*
cvs*Dialog.Label.font:     -*-helvetica-medium-o-normal--*-*-240-100-100-*-*-*-*-*
cvs*Help*font:             -*-times-medium-i-*-*-*-100-100-100-*-*-*-*
cvs*History*font:          -*-times-medium-i-*-*-*-120-100-100-*-*-*-*
cvs*plotForm.testForm*font: -*-helvetica-medium-r-*-*-*-80-*-*-*-*-*-*
cvs*plotForm*Form.Text*font:       -*-times-medium-i-*-*-*-120-100-100-*-*-*-*
cvs*plotForm*Form.Command.font:  -*-times-medium-i-*-*-*-100-100-100-*-*-*-*
cvs*plotForm*Label.font:  -*-times-medium-r-*-*-*-100-100-100-*-*-*-*
cvs.Plot.Command.font:     -*-times-bold-i-*-*-*-100-100-100-*-*-*-*
cvs.Plot.Label.font:       -*-times-medium-r-*-*-*-100-100-100-*-*-*-*
cvs*Command.font:          -*-times-medium-i-*-*-*-100-100-100-*-*-*-*
cvs*font:                  -*-times-medium-r-*-*-*-100-100-100-*-*-*-*
cvs*smallFont:     -schumacher-clean-medium-r-normal--6-60-75-75-c-40-iso8859-1

#else must be RELEASE 2 or 3
cvs*commandButton.font:    -*-helvetica-bold-r-normal--*-*-140-*-*-*-*-*-*-*
cvs*Dialog.Command.font:   -*-helvetica-medium-o-normal--*-*-240-*-*-*-*-*-*-*
cvs*Dialog.Label.font:     -*-helvetica-medium-o-normal--*-*-240-*-*-*-*-*-*-*
cvs*Help*font:             -*-times-medium-i-*-*-*-120-*-*-*-*-*-*
cvs*History*font:          -*-times-medium-i-*-*-*-140-*-*-*-*-*-*
cvs*plotForm.testForm.Toggle.font: -*-helvetica-medium-r-*-*-*-80-*-*-*-*-*-*
cvs*plotForm*Text*font:    -*-times-medium-i-*-*-*-140-*-*-*-*-*-*
cvs*plotForm*Command.font: -*-times-medium-i-*-*-*-140-*-*-*-*-*-*
cvs*plotForm*Label.font:   -*-times-medium-r-*-*-*-140-*-*-*-*-*-*
cvs*Plot.Command.font:     -*-times-bold-i-*-*-*-140-*-*-*-*-*-*
cvs*Plot.Label.font:       -*-times-medium-r-*-*-*-140-*-*-*-*-*-*
cvs*Command.font:          -*-times-medium-r-*-*-*-120-*-*-*-*-*-*
cvs*Label.font:            -*-times-medium-i-*-*-*-120-*-*-*-*-*-*
cvs*font:                  -*-times-medium-r-*-*-*-80-*-*-*-*-*-*
cvs*smallFont:             6x10
#endif
```

help.txt

```
Welcome to the Cardiovascular Simulator!
This program requires the X Window System's window manager.

A FEW POINTERS:

  Use the LEFT MOUSE BUTTON to click on command buttons on the screen.

  To start viewing catheterization data, you must make a NEW PLOT.
  Identify values of points on plots by moving the mouse pointer.      Click to make a mark.

  For DELETE and PRINT, the mouse pointer changes to a cross-hair; click on an area of the screen.
  The DELETE button may be used only for removing simulator windows.

  mwm:   To MOVE, ICONIFY, and RESIZE windows, hold down the left mouse button and drag on window edges.

  When creating new windows, holding down the left mouse button and dragging alters the window size.

  The variable choice diagram for New Plot contains both independent and dependent variables.
  The variables above the dotted line are the dynamically-changing dependent variables, whereas those
  below the line are altered only by the user or the baroreceptor control system.

Simulator Button Definitions:
      NEW PLOT: Look at a plot of one variable against others.
      SET PARAMETER: Choose a simulation parameter to create a modification number line.
      PARAMETER LIST: Obtain a list of current simulation parameter values.
      STOP/RESUME: Change the execution status of the simulation.
        RESUME TRANSIENT starts immediately, STEADY STATE discards data until steady state is reached.
      BAROREFLEX: Activate the experimental baroreceptor feedback system.
      PATIENTS: Choose from a list of patient cases.      Use RESET to get back to the default simulation.
      PRINT: Choose the screen or a window for printout at a PostScript printer.
      DELETE: Choose a window (like this one) to delete from the cardiovascular simulator.
      RESET: Reset simulator to pristine experiment on a normal subject.
      QUIT:  Stop everything and exit the simulator.
      HELP: Create a window containing this page.

GOOD LUCK and HAVE FUN!
```

convergence.failure

```
SIMULATION HALTED for failure to converge.

You must RESET the simulation or change some
parameters and press the Stop/Start button twice,
asking for steady state solution to continue
the simulation.      Click in these windows to remove
them.
```

Patients/hist.1

```
### Patient 1 History ###
Patient 1:
    This patient is 49 years old, is well developed, and presents
with dyspnea and chest pain on exertion, and frequent ventricular
arrhythmias.  He has had several unexplained syncopal attacks, from
which he has recovered without incident.  Physical exam revealed a
prominent LV impulse, a palpable fourth heart sound, a biphasic
carotid pulse with a sharp rise time, and a systolic ejection murmur
which increased in intensity with inspol.

### Patient 1 Parameters ###
venous sprvolume   2000.0
system volume      6000.0
lv resistance      0.1
lv cdiastolic      3.0
lv csystolic       0.1
rv cdiastolic      4.0
system hr          100.0
```

Patients/hist.2

```
### History ##########################################################
Patient 2:
    This patient is a 24 year old woman who presents with a long
history of severe dyspnea and now requires constant oxygen
supplements.  She also has had a long history of peripheral edema.
Physical exam reveals elevated jugular venous pressure, a prominent
cardiac impulse in the subxyphoid area, and a loud second heart
sound which was not split.  She had obvious cyanosis and clubbing of
the fingers.

### Data
pa capacitance     2.5
system volume      6000
pulmo resistance   1.0
rv cdiastolic      10.0
rv csystolic       0.4
system hr          120.0
```

Patients/hist.3

```
### History ##########################################################
Patient 3:

    This patient is a 67 year old man who complained of awareness
of a rapid pulse rate, and a forceful heart beat.  In addition, he reported
that his exercise tolerance had decreased, and that he would get short of
breath and fatigued more easily than ever before.  He had no previous history of
heart disease.  A previous physical examination done a year ago had discovered
a pulsatile abdominal mass, with an audible bruit in its vicinity.  He had
had good femoral pulses bilaterally at that time.  Examination now confirms the
presence of the pulsatile mass and bruit.  The cardiac exam reveals a rapid
heart rate with a very forceful apical impulse.  There was a grade 2/6 systolic
murmur heard at the apex and over the second intercostal space to the left of
the sternum.  No diastolic murmurs were heard.  Prominent capillary pulsations
were noted in the nail beds and under the tongue.  Femoral pulses were present
bilaterally, although the left was weaker than the right.  The patient was taken
to the catheterization lab for confirmation of the diagnosis.

### Data
microvascular resistance 0.1
venous capacitance 70
venous sprvolume   1500
pulmo resistance 0.04
lv csystolic       0.2
rv csystolic       0.6
system volume      5500
system hr          120.0
```

Patients/hist.4

```
### History ##########################################################
Patient 4:
    This patient was a 41 year old mother of three children who was
well (except for the usual colds, etc.) until last year when she began
to experience shortness of breath on even minor exertion, and episodes
of shortness of breath during the night which were relieved by standing
up.  She denied any chest pain, chest infections, palpitations; but did
report some mild swelling of the ankles.  Her physical activities have
become severely limited because of the shortness of breath.  She has
been taking several medications at the request of her physician.  She
comes to you for hemodynamic evaluation and assessment.  Her exam
reveals an enlarged heart, regular rhythm, an S-3 gallop, no murmurs,
and fine rales at both lung bases.  There was 2+ ankle and pretibial
edema.

### Data
micro resistance 0.9
arterial capacitance 1.0
venous sprvolume 2200.0
pottery capacitance 2.5
pulmo resistance 0.140
lv sprvolume 50.0
lv cdiastolic 5.0
lv csystolic 1.2
rv sprvolume 50.0
rv cdiastolic 11.0
rv csystolic 3.6
system volume 6000.0
system hr 125.0
```

Patients/hist.5

```
### History ##########################################################
Patient #5:
    The patient is a 60 year old man who entered the hospital with severe
crushing substernal chest pain of two hours duration.  He appears short of
breath.  He is diaphoretic, has cold clammy skin, and his nail-beds are
cyanotic.  He is very lethargic and seems somewhat confused.  He has low blood
pressure, elevated jugular venous pressure, a regular cardiac rhythm with
an S-3 gallop, and examination of the lungs reveals bibasilar fine rales.
He has no edema.  His urine output seems to be very scanty.

### Data
micro resistance 1.4
arterial capacitance 0.5
venous sprvolume 1000
pottery capacitance 2.0
lv sprvolume 25
```

lv odiastolic 8
lv systolic 2.5
rv spvolume 25
rv odiastolic 15
rv systolic 4
system hr 110

Patients/hist.6

86 History ##
Patient 6:

This 55 year old male bank vice-president had been in excellent health until one month ago when he began to notice unusual shortness of breath and substernal "*pressure*" which was associated with physical exercise. The *symptoms* would be brought on quite predictably after carrying a heavy briefcase up to the third floor of the parking garage, for example. Rest would always relieve the discomfort after a few minutes.

He had a known heart murmur which was first noted on a routine physical examination ten years ago, but at the time it had been considered to be an "innocent" murmur.

On the day of admission he had been carrying a bag of cement to his backyard when he suddenly became lightheaded and then lost consciousness.

86 Data
arterial capacitance 0.8
venous spvolume 2000
lv resistance 0.1
lv odiastolic 8
lv systolic 0.3
system volume 5100
system hr 60

Patients/mkpat

crypt 'crypt keyword < ../../cvLib/Patients/key.cr' < $1 > ../../cvLib/Patients/$1