1

UNIVERSAL ELECTRONICS, INC. 88
MICRO
ELECTRONICS
1000 10th Avenue, Suite 100, San Francisco, CA 94103

**National Library
of Canada**

**Bibliothèque nationale
du Canada**

Canadian Theses Service

Service des thèses canadiennes

**Ottawa, Canada
K1A 0N4**

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

**A Computer-Driven
Astronomical Telescope Guidance and Control System
with
Superimposed Star Field and Celestial Coordinate
Graphics Display**

by

Douglas B. George, B.A.Sc.

**A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of
Master of Engineering**

**Ottawa-Carleton Institute for Electrical Engineering
Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University**

October, 1987

**(c) copyright
1987, Douglas B. George**

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-51157-5

**The undersigned hereby recommends to the Faculty of Graduate
Studies and Research acceptance of the thesis,
A Computer-Driven
Astronomical Telescope Guidance and Control System
with
Superimposed Star Field and Celestial Coordinate
Graphics Display
submitted by Douglas B. George, B.A.Sc., in partial fulfillment of
the requirements for the degree of Master of Engineering**

Thesis Supervisor

Chairman,
Department of Systems and
Computer Engineering

**Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University
October, 1987**

I. ABSTRACT

A Telescope Graphics Display System (TGDS) was proposed which fully integrates a computer into the operation of an astronomical telescope. A prototype was constructed to demonstrate the feasibility and utility of two key novel features: (1) the real-time display of star fields in the vicinity of the telescope field of view, and (2) the real-time display of symbols representing celestial objects, and other data, superimposed upon the actual images in the telescope eyepiece. To our knowledge, this is the first system proposed to perform these functions. A variety of applications are proposed, ranging from training observers to performing measurements. The prototype was successfully tested under actual observing conditions.

II. ACKNOWLEDGEMENTS

I would like to thank my thesis supervisor, Dr. L. R. Morris, for his assistance, encouragement, and enthusiasm. Dr. Morris originated the idea of using a small video monitor as a "heads-up" eyepiece display - one of the primary components of this thesis - and suggested some of the applications of such a device.

I would also like to thank Dr. D. L. Welch of the Dominion Astrophysical Observatory for supplying a set of computer databases.

III. TABLE OF CONTENTS

<u>Section</u>	<u>Title</u>	<u>Page</u>
I.	ABSTRACT	i
II.	ACKNOWLEDGEMENTS	ii
III.	TABLE OF CONTENTS	iii
IV.	LIST OF FIGURES	vi
V.	LIST OF PLATES	vii
VI.	LIST OF APPENDICES	viii
1.	INTRODUCTION	1
2.	OVERVIEW	3
2.1	GENERAL TELESCOPE OPERATION	3
2.2	TELESCOPE GRAPHICS DISPLAY SYSTEM	5
3.	EXISTING TECHNOLOGY	12
3.1	REAL-TIME OBSERVING	12
3.2	DATA REDUCTION	13
3.3	OBSERVATION PLANNING	13
3.4	EQUIPMENT DESIGN	14
3.5	SUMMARY	14
4.	APPLICATIONS	15
4.1	OBSERVING AID	15
4.1.1	Position Measurement	15
4.1.2	Location Aid	16
4.1.3	Object Identification	17
4.1.4	Tracking Moving Objects	17
4.1.5	Event Timing	18
4.1.6	Intensity Measurement	18
4.1.7	Auxiliary Functions	19
4.2	TRAINING	20
4.2.1	Classroom Demonstration	20

4.2.2	Indoor Practice	21
4.2.3	Field Operation	21
4.3	SUMMARY	22
5.	PROTOTYPE SYSTEM OVERVIEW	23
5.1	PROTOTYPE SPECIFICATIONS	23
5.2	SYSTEM HARDWARE	26
5.2.1	Telescope	26
5.2.2	Optical Assembly	29
5.2.3	Computer	29
5.2.4	Motor Controller	33
5.3	OPERATION AND USER INTERFACE	35
5.3.1	Display	35
5.3.2	Keyboard Control	35
6.	HARDWARE DESIGN	39
6.1	OPTICAL ASSEMBLY	39
6.1.1	Mechanical Design	39
6.1.2	Optical Design	43
6.2	MOTOR CONTROLLER	46
6.3	VIDEO CONTROLLER	51
7.	MAIN COMPUTER SOFTWARE DESIGN	52
7.1	DATABASE MANAGEMENT	52
7.1.1	Database Storage	53
7.1.2	Pre-Processing	54
7.2	GRAPHICS GENERATION	55
7.2.1	Projection	55
7.2.2	Real-Time Display	56
7.2.3	Screen Interface	57
7.3	TELESCOPE CONTROL	59
7.4	PROGRAM DESCRIPTION	60
8.	MOTOR CONTROLLER SOFTWARE DESIGN	64
8.1	MOTOR CONTROL	64

8.2	TIMERS	65
8.2.1	Sidereal Timer	65
8.2.2	Motor Timer	65
8.3	BACKLASH CONTROL	67
8.4	PROGRAM DESCRIPTION	68
9.	RESULTS	72
9.1	PROTOTYPE SYSTEM OPERATION	72
9.2	QUALITATIVE TESTS	72
9.3	SYSTEM PERFORMANCE TESTS	74
9.4	A TYPICAL OBSERVING SESSION	76
10.	CONCLUSIONS	82

IV. LIST OF FIGURES

<u>Number</u>	<u>Title</u>	<u>Page</u>
2.1-1	Astronomical Telescope	4
2.1-2	Celestial Coordinate System	6
2.2-1	Telescope Graphics Display System	7
2.2-2	Observing Aids	9
2.2-3	The TGDS as an Observing Aid	10
5.2-1	Major System Components	27
5.2.1-1	Telescope Optical System	30
6.1.1-1	Optical Assembly Design	40
6.2-1	Motor Controller Power Supply	48
6.2-2	Motor Controller Microcomputer	49
6.2-3	Stepper Motor Drivers	50
7.4-1	Main Program Structure Chart	61
8.4-1	Motor Controller Structure Chart	69

V. LIST OF PLATES

<u>Number</u>	<u>Title</u>	<u>Page</u>
5.2.1-1	Celestron-8 Telescope	28
5.2.2-1	Optical Assembly	31
5.2.3-1	Main Computer	32
5.2.4-1	Motor Controller	34
6.1.1-1	Optical Assembly and Telescope	42
6.2-1	Motor Controller Electronics	47
9.1-1	Complete TGDS System	73
9.3-1	Initial Display	78
9.3-2	Centering Betelgeuse	78
9.3-3	Switching to Telescope View Mode	79
9.3-4	Centering the Pleiades Cluster	79
9.3-5	Larger View of Pleiades	80
9.3-6	Entering Designation of Target Star	80
9.3-7	Star Centered	81
9.3-8	Telescope View of SAO 76136	81

VI. LIST OF APPENDICES

<u>Number</u>	<u>Title</u>	<u>Page</u>
A.	CALCULATIONS	84
A.1	CYLINDRICAL PROJECTION	84
A.2	LAMBERT CONFORMAL CONIC PROJECTION	85
A.3	ALTAZIMUTH ROTATION	87
A.4	MOTOR STEP SIZE	88
A.5	SIDEREAL TIME	89
A.6	COORDINATE PRECESSION	90
B.	REFERENCES	91
C.	BIBLIOGRAPHY	93
D.	SOFTWARE LISTINGS	95

VII. GLOSSARY

Asteroid	A solar-system body composed of rock and iron but much smaller than a planet, in orbit around the sun.
Catalog	A table of data on stars, nebulae, galaxies, or other objects.
Celestial Coordinates	The coordinate system used for measuring the position of stars and other celestial bodies. See Right Ascension and Declination.
CCD	Charge-Coupled Device: a type of imaging device.
Clock Drive	A drive motor arrangement which keeps a telescope aligned on the stars as the Earth rotates.
Declination	The equivalent of Latitude in the celestial coordinate system.
EGA	Extended Graphics Adapter: an IBM-defined standard video display system for personal computers.
Equatorial	A mounting or coordinate system with one axis parallel to the Earth's poles.
Equinox	The point in time when the sun is directly over the equator.
First Point of Aries	The reference point for the celestial coordinate system, in the direction of the sun at the spring equinox.
ISR	Interrupt Service Routine: software used to service interrupts.
Lambert Conformal Conic	A variation of the conic projection used in drawing maps.

Magnitude	The unit of measurement of the apparent brightness of a star as seen from Earth. The scale is logarithmic so that a magnitude -1 star is 100 times brighter than a magnitude 4 star.
NTSC	The North American standard for broadcast television, devised by the National Television Standards Committee.
Occultation	The passage of a solar system body in front of another object, such as a star.
Photometer	A device used to measure the magnitude of a star.
Planisphere	A simple star map with moving parts which can be adjusted to show the sky as it appears at any date and time.
Plossl	A type of wide-field eyepiece.
Precession	The movement of the Earth's axis relative to an inertial frame.
Reticule	A set of crosshairs or calibrated lines which appear in an eyepiece.
Right Ascension	The equivalent of Longitude in the celestial coordinate system.
RS-232	A standard for serial data communications.
SAO Catalog	Smithsonian Astrophysical Observatory Catalog.
Schmidt-Cassegrain	A type of optical system containing both lens and mirror elements.
Setting Circles	Devices for measuring the angular position of a telescope in Right Ascension and Declination.

Sidereal Time	Time measured in terms of the Earth's rotation relative to the stars.
TGDS	Telescope Graphics Display System.
YBSC	Yale Bright Star Catalog.
VGA	Video Graphics Adapter; an IBM-defined standard video display system for personal computers.

1. INTRODUCTION

In recent years computers have come into widespread use in astronomical research. Applications range from sophisticated data reduction to instrument control and data collection. Large observatories have sophisticated systems for controlling the telescope and collecting data; in most cases, the smaller institutions do not.

Research is now progressing into methods for making smaller observatories more productive and scientifically useful. One particularly successful concept is the Automatic Photoelectric Telescope which can automatically perform complex variable star observations without any human intervention [1]. Such systems, while very attractive, are only practical in areas where excellent weather conditions prevail, and can only make certain types of observations. They are also extremely expensive. Therefore, such fully automatic systems are not suited to all observatories.

The chief aim of the project was to devise a telescope "operating system" which assists, rather than eliminates, the observer. The novel concept envisioned was a graphics display superimposed on the image formed by the telescope – a kind of "heads-up-display" for a telescope. This Telescope Graphics Display System (TGDS) would provide information on the objects being observed and assist in the performance of measurements.

The TGDS was intended to be used on smaller instruments such as would be found in smaller university or serious amateurs' observatories. Typically, a variety of types of observations are performed with one instrument; therefore a flexible system is desirable.

Since not all possible applications could be accommodated in the time available, it was decided to pursue a more limited goal of designing and building a proof of concept system. Therefore, a system was designed which performed the most important basic

function: the location and identification of stars and other objects. There are significant challenges in implementing this basic function, since it incorporates all of the aspects of telescope control, database management, and graphics generation.

The proposed limited system would still have a number of applications, including: training observers, locating and identifying astronomical objects, and providing a very convenient user-interface for the telescope. Many additional applications could be accommodated with relatively little extra work.

The basic idea of the heads-up display, and the application to locating and identifying objects in the telescope, was proposed by Dr. L. R. Morris. The original contributions of the author include: (a) the applications described in Section 4, with the above exception, (b) the design and construction of the optical system and motor control electronics described in Section 6, (c) the design of the user interface as described in Section 7, and (d) the development and testing of the main computer (8088) and motor controller (6809) software as described in Sections 7 and 8.

In the following sections, the TGDS is described in detail. Section 2 describes the operation of telescopes in general, and how the TGDS operates. Existing applications of computers to astronomy are discussed in Section 3. Section 4 describes a number of applications which were identified. In Section 5, the specifications of a prototype TGDS are developed, and the components and operation of a system designed to those specifications are described. The design of new hardware constructed for the system, and the selection of the required video display card, is described in Section 6. The design of the main computer and motor controller software is described in Sections 7 and 8, respectively. The results of tests on system performance are described in Section 9, and conclusions are included in Section 10.

2. OVERVIEW

In this section, the general features of modern telescopes are discussed. The Telescope Graphics Display System is then described in terms of its operation with a telescope.

2.1 GENERAL TELESCOPE OPERATION

A typical telescope is shown in Fig. 2.1-1. There are two essential parts to any telescope: the optics and the mounting.

The optics on modern telescopes generally consist of extremely accurate first-surface mirrors. The mirrors are constructed of high quality Pyrex or Cervit glass, with a reflective coating of aluminum or silver (with protective overcoatings). The performance of a telescope is characterized by its aperture, which corresponds to the diameter of the primary mirror (or lens). The larger the primary mirror, the more light is collected. Resolution also depends on the aperture, up to the limit due to atmospheric turbulence.

Light is brought to a focus outside the instrument, where an eyepiece, camera, or other instrument may be mounted. Focussing is done by varying the distance between the primary mirror and the eyepiece.

The mounting may be in one of several configurations. Most telescopes designs utilize some variation of the Equatorial mounting. Such mountings have a Polar or Right Ascension axis carefully aligned to be parallel with the Earth's poles, and a perpendicular Declination axis. The terms Right Ascension and Declination refer to the equivalents of Longitude and Latitude in the celestial coordinate system.

Fig. 2.1-1 Astronomical Telescope

The position of the telescope can be determined directly by measuring the angular position of the telescope axes. Declination is usually measured in degrees north or south of the celestial equator. Right Ascension is usually measured in hours, minutes, and seconds east of the First Point of Aries. The First Point of Aries is the prime meridian for the celestial coordinate system, corresponding to the direction of the sun at the spring equinox [2].

Due to the Earth's rotation, the stars appear to move around the Earth's axis over time. With the Right Ascension axis parallel to the poles, it is a simple matter to slowly rotate the telescope to counteract this motion. As shown in Fig. 2.1-2 the rotation angle of the Earth is measured from the First Point of Aries to the meridian. This number, when measured in hours, minutes, and seconds corresponds to the instantaneous Right Ascension of the Meridian, and is known as Local Sidereal Time. The Hour Angle of an object is equal to the Sidereal Time minus its Right Ascension; this quantity varies with time.

2.2 TELESCOPE GRAPHICS DISPLAY SYSTEM

The proposed Telescope Graphics Display System connects to an equatorially-mounted telescope as shown in Fig. 2.2-1. The key elements of the system are a master computer, motor controller, and an optical assembly.

The master computer controls the entire system. It controls the position of the telescope through the motor controller, and is thus able to accurately determine the telescope position. It also has a relatively high-resolution graphics display and a large database of astronomical objects.

The motor controller moves the telescope using two stepper motors, one on each axis of the instrument. The telescope position is determined either by counting steps (open loop configuration), or by using an optical shaft encoder (closed loop configuration) and interpolating by counting steps.

Hour Angle = $\widehat{QM}$
 Right Ascension = $\widehat{VQ}$
 Declination = $\widehat{PQ}$
 Siderial Time = Hour Angle of V
 = $\widehat{VM}$

Fig. 2.1-2 Celestial Coordinate System

Fig. 2.2-1 Telescope Graphics Display System

The **optical assembly** superimposes computer-generated graphics information in the eyepiece of the telescope. This is done through combining two light paths with a partially-aluminized mirror.

The TGDS thus incorporates two major innovations. The first innovation is the use of a computer database and graphics system to display information in real time. For example, a computer-generated star map can be synchronized to the telescope position. The user may command the computer to scroll star maps on the screen, with the computer simultaneously moving the telescope. Alternatively, by sensing the position of the telescope, the computer may scroll the star maps as the user moves the telescope manually. Thus, a fast, user-friendly interface simplifies the operation of the telescope. The detailed star field displays eliminate the need for handling large, awkward star maps.

The relationship of the TGDS to other observing aids is shown in Fig. 2.2-2. The time-honoured method of locating an object is to use a paper star chart or catalog, and to manually adjust the telescope using either setting circles or using a wider-field finder telescope (Diagram A). A variation on this is the Planisphere which operates as small, flat planetarium, allowing the user to rotate the chart through a window to see the sky as it appears at a particular latitude at any date and time (Diagram B). A recent innovation is the use of a computer program, such as "Planetarium" [3] to display or plot star maps (Diagram C). Similar programs are used by professional astronomers to prepare special-purpose star maps when planning observations. Within the last two years, small micro-controllers with star and object databases have become commercially available (Diagram D). These allow the user to enter an object's name or designation, and the telescope automatically slews to the desired location.

The TGDS is a logical extension of this progression, as shown in Fig. 2.2-3. The TGDS is basically a combination of Diag. C and D of Fig. 2.2-2. The system allows the user to interact in real-time with a computer system which controls the telescope, and which generates detailed video graphics star maps synchronized with the telescope position.

Fig. 2.2-2 Observing Aids

Fig. 2.2-3 The TGDS as an Observing Aid

The second major innovation is the superimposition of graphics in the eyepiece. This allows the system to (a) annotate actual celestial objects viewed directly in the eyepiece, (b) aid in measurements, (c) locate and identify celestial objects, (d) track moving objects, (e) aid in event timing, (f) aid in intensity measurements, and (g) allow the operation of the telescope for training while indoors, by simulating the sky image in the telescope eyepiece. This expands the utility of the TGDS from a telescope accessory to a complete system for performing scientific observations and training observers.

3. EXISTING TECHNOLOGY

Before proceeding with the development of system, it was important to first explore systems already in existence. This aided in the evaluation of the proposed system in terms of new and existing applications.

A large variety of applications of microcomputers in astronomy have been identified. Every large professional observatory uses computers extensively for telescope and instrument control, data logging, and other tasks. The applications of interest here would be those suitable to smaller university or amateur observatories.

Applications for amateur observatories were discussed in detail in a paper by Manly [4]. He subdivided applications as follows: real-time observing, data reduction, observation planning, and equipment design.

3.1 REAL-TIME OBSERVING

Real-time observing applications can be subdivided into three classes: **prompting programs, telescope control, and automated instrument control.**

Prompting programs do not control any instruments, rather they prompt the observer to follow a pre-determined observing program. This is useful for reminding observers of checklists and complicated observing sequences. It eliminates wastage of time and ensures that the observing session is carefully planned.

Telescope control is an obvious application of computers. Applications range from simple pointing control to advanced systems which model mounting flexure and gear train errors for higher accuracy.

Automated instrument control includes data logging, as well as control of mechanical elements such as aperture and filter wheels. Other instrument control

applications include monitoring and control of instrument parameters such as setting gain, bandwidth, or supply voltages.

Some systems have been designed to provide automatic photograph tracking. The most advanced systems are the fully automated Automatic Photoelectric Telescopes developed at the Fairbourne Observatory [5]. These computerized instruments open the roof, align to the sky, and make complex observations without any human intervention. At present, only a few such telescopes exist in the world.

3.2 DATA REDUCTION

Microcomputers can be used to record and reduce observation data. For example, a large amount of calibration and manipulation of star intensity data is required to produce a light curve for a variable star [6]. Accumulated databases of such information can be used to aid in detection of systematic errors and variations from observer to observer.

Much more sophisticated image processing software is in use at large institutions which use Charge-Coupled Device imagers. As these CCDs become more available, smaller institutions will likely begin using these techniques.

3.3 OBSERVATION PLANNING

Observation of specific stars and other objects requires careful preparation so that the correct object can be identified. The preparation of special-purpose star maps by a computer is now a widely-used technique.

Another application is the prediction of astronomical events, such as eclipses, occultations, and the positions of solar system objects. As an example, an occultation is the passage of a planet, moon, or asteroid in front of another body such as a star. In order to predict observable events, massive database searches must be performed using extremely precise orbit calculations. The results of such observations gives extremely precise data on the size, shape, and orbit of solar system objects [7].

3.4 EQUIPMENT DESIGN

An application not directly related to observing is the design of astronomical equipment. The major application in this area is optical ray-trace programs used in the design of new eyepieces and telescopes. Extensive work is being done in this area in both the amateur and professional domain.

3.5 SUMMARY

As can be seen, microcomputers are presently being used in a growing number of applications in astronomy.

A survey of the literature in this field has not identified any previous attempts to build a graphics display system similar to that described in this thesis. Large observatories systems which perform imaging using CCDs can display graphics information superimposed on acquired digital images, but this is not done in real time. The CCD and associated display technology is also expensive and therefore generally not available to smaller observatories.

This survey suggested that the proposed Telescope Graphics Display System is indeed a unique and potentially very useful tool for astronomical research. The key to making this tool useful, however, is the identification of specific application areas.

4. APPLICATIONS

The proposed Telescope Graphics Display System can be used in a variety of applications. The system can be used as a training device, or as a general purpose observing aid. As an observing aid, the TGDS could assist in the performance of measurements, locating objects, identifying objects, photographing moving objects, and a variety of other functions. As a training tool, it could provide a realistic indoor "sky simulation", and a smooth transition to actual observing conditions.

4.1 OBSERVING AID

The TGDS can be used as a tool during regular observing sessions. In addition to providing a "user-friendly" interface to the telescope, applications include **position measurement, locating objects, identifying objects, tracking moving objects, timing events, intensity measurement, and auxiliary observing aids.**

4.1.1 Position Measurement

Several types of measurements can be made using a small telescope. For example, the measurement of separation and position angle of double stars can be performed. Such observations are presently done using several techniques, many of which have poor accuracy. These include sketching, using a micrometer, and the drift method (timing the drift of the object past a cross-hair with the telescope clock drive off).

Using the eyepiece graphics system, the operator could align two computer generated cross-hairs on different points in the sky. The computer would ensure that they appear motionless with respect to the stars even as the telescope moves, and could calculate their exact position angle and displacement. This would reduce bias and error, as well as permit the measurement of distances larger than one eyepiece field. Two-dimensional measurements could also be made much more easily. An application of this

could be the measurement of the position and direction and speed of motion of a newly-discovered object.

4.1.2 Location Aid

One technique for locating objects is to use their celestial coordinates, as given in a table. This is often not sufficient, however, to easily identify a particular faint star in a crowded field. The alternative, "star-hopping" from an easily located bright object following a star chart, can be time consuming and difficult. Thus, a fair degree of skill can be required to make what are otherwise fairly simple observations.

The graphics display could be used to locate and identify objects of interest. The system would first be initialized using a bright, easily located star (this need only be done once per observing session). The observer would then center a faint object on the computer screen, and the computer would automatically slew to the object using the stored coordinates. The eyepiece graphics display could be very useful in crowded star fields, in that it could annotate the desired star in the display. This could be especially useful if the telescope slewing accuracy is limited, since the observer can adjust the graphics star pattern to match the actual star pattern. This would ensure accurate positioning and correct star identification.

If, for example, the user was interested in observing a particular variable star, the system could identify the star, and label several nearby fixed comparison stars as well. The system could then slew between the stars on request to permit rapid comparison of their intensities. Information on the comparison stars could also be displayed in the eyepiece, such as their catalog designation and visual magnitude. The acquired data and time of each observation could be stored by the system, either using keyboard input or by directly reading the measuring instrument.

This same technique could be applied to asteroidal observations. The computer could calculate the exact position of the object at any time, and indicate this on the display. This would greatly speed up the task of identifying the correct object, and reduce the possibility of error.

This method could also be used to time re-appearances of stars from behind the moon during an occultation. Since the star cannot be seen until after the re-appearance event, the event is often missed due to a mis-aligned telescope. The exact position of the telescope could be checked a few minutes before the event, and an indicator could show the expected location of re-appearance.

4.1.3 Object Identification

Many amateur astronomers regularly patrol the skies looking for new objects such as comets, asteroids, novae, and supernovae. A heads-up display system with a large object catalog could annotate the known objects, in order to distinguish them from new objects.

Once a new object is discovered, the measurement functions could be used to accurately determine the position of the object. The observer would center a computer-generated cross-hair display on the object, and the computer would automatically calculate the position. By repeating this a short time later, the computer could calculate the direction and speed of motion of the object. The time of each observation would also need to be stored for this purpose.

4.1.4 Tracking Moving Objects

Astrophotography typically involves long time exposures. In order to ensure correct tracking, the instrument must be manually guided to prevent trailing of the image. This involves using a cross-hair eyepiece aligned on a nearby star. In the case of objects which move quickly with respect to the background stars, such as comets, guiding can be very difficult, or even impossible.

One method for photographing these objects is to align a reticule on a fixed star. This reticule is then carefully adjusted at regular intervals in an opposite direction to the motion of the object. If this moving reticule is kept aligned on the fixed star, the telescope will be slowly moved to track the object [8].

Since the computer can easily calculate the expected motion of the object, and can generate an image of a reticule in the eyepiece, it can perform the tedious task of accurately adjusting the reticule position.

4.1.5 Event Timing

A potential application of the TGDS is the timing of events, such as the appearance and disappearance of a star during an occultation. Ideally this could be done using a high-speed, high-sensitivity photometer, but these can be difficult to construct. A simpler approach uses the eyepiece display as a timing mechanism.

The proposed method of operation is similar to the "eye-and-ear" approach, in which the observer attempts to estimate tenths of seconds between successive tones from a time signal. This method virtually eliminates the observers' response time from the measurement. The difficulty lies in estimating the tenths of seconds.

With the graphics system, ten bars could be illuminated in sequence in the eyepiece to indicate tenths of seconds. They could be grouped in two sets of five so that the pattern of lights could be easily read and remembered. The observer need only remember the appearance of the image at the instant the event occurs; the number of illuminated bars indicates the exact fraction of a second at which the event occurred.

A simpler, very portable version of the graphics display could be easily built. This could help improve the accuracy of visual observations without the expense of a high-speed photometer.

4.1.6 Intensity Measurement

One application of the TGDS could be the measurement of the magnitudes of stars and other objects. The procedure would be for the observer to adjust the intensities of three artificial star images until they matched the intensities of a target star and two comparison stars. With the two comparison stars bracketing the target star in magnitude, the computer could interpolate the magnitude of the target star. This would have several

advantages for visual magnitude estimates, including reduction of bias, and an increase in precision. This would provide an improvement in quality of observations on telescopes not equipped with photometer equipment. This technique may also be easier to use than photometer equipment, although probably not as accurate. It would, of course, necessitate careful calibration of the display intensities, and would probably require a large range of grey scale values from the display hardware.

An interesting possibility is the measurement of the integrated magnitude of diffuse objects such as comets. At present, this is quite difficult to do because the large objects do not fit into the small diaphragms of most photometers. If the diaphragms are made larger, sky background and nearby stars cause further difficulties. Visual observations are also difficult because comparing a diffuse object to a discrete source, such as a comparison star, is extremely difficult. A typical method, although awkward, is to de-focus the telescope until the comparison star images appear roughly the same size. The TGDS, however, could generate a graphical image of a diffuse object with a known integrated magnitude. The size, condensation and intensity of the image could be adjusted by the observer until it matched the comet, then an accurate intensity value could be read out.

4.1.7 Auxiliary Functions

Other functions could be added which would be ancillary to the main display function. Various parameters could be displayed, including the telescope's present position, operating mode, and sidereal, local or universal time. The compass directions could be indicated in the display.

Other utility programs, such as data logging or observing sequence prompting programs could also be added.

4.2 TRAINING

Learning to use a telescope can be a daunting problem for both the novice amateur and the student. While at first glance the operation of a telescope may seem fairly simple, it is in fact very difficult to locate interesting objects among a sea of faint, unfamiliar stars. As a result, many beginners are frustrated and disappointed with their first experiences.

Some computer-based devices presently on the market allow the user to slew the telescope automatically from one object to another, using a small database. This overcomes the difficulty in locating some objects, but does not encourage the development of observing skills. Without these skills, the ability to locate objects stops at the end of the database.

There are three modes in which the TGDS could be used: classroom demonstration, indoor practice, and field operation.

4.2.1 Classroom Demonstration

In this mode, the telescope eyepiece display would be replaced by a standard video monitor. The monitor could display views ranging from "star-map" size to actual telescope field size. Control of the "telescope" position would be via the keyboard or a joystick. An instructor could use this to demonstrate basic observing procedures to a small group of students.

Techniques such as star-hopping and the use of coordinate setting circles could be demonstrated. The relationship between the field of view seen in the telescope and on stars maps could be shown. The conditions of typical events could be simulated, and observation techniques demonstrated. For example, variable star observations could be simulated, complete with standard comparison stars and an artificial variable star.

Although this is possible at present using such programs as "Planetarium" [3], the advantage of the TGDS is the smooth transition to the indoor practice and field operation

modes of operation. In addition, existing software packages do not provide realistic representations of the night sky, and are either limited to naked-eye visible stars, or cannot operate in real time. Existing software packages were designed to use older graphics cards, and could not make use of the grey scale capabilities of the newer cards. They therefore represent brighter stars as larger circles, with generally very poor results. Existing packages were not designed with the real-time constraints of the TGDS. Rapid and accurate display of faint stars is inherent to the primary concepts of the TGDS.

4.2.2 Indoor Practice

Using the TGDS eyepiece display would permit a real-time simulation of the night-time observing environment indoors, using the actual telescope under normal lighting conditions. Thus, the TGDS effectively functions as a **realistic, real-time night sky simulator**.

With the TGDS simulating the appearance of the sky, as seen through the telescope eyepiece, the student could practice operating the telescope. A tutorial manual or special software could be used to teach techniques such as star-hopping, star identification, and various types of observations. The student could become familiar with the telescope and its operation under comfortable conditions, and attempt simulated observing runs (with known event data for testing purposes).

This technique would provide a means for evaluating an observer's proficiency and accuracy in performing various types of observations. The ability of the system to generate accurate simulations of various events could also be used by experienced observers in preparing for difficult observing runs.

4.2.3 Field Operation

Once operating in the field, the student could use the system to aid in locating objects. Upon request, the display could show telescope position, and identify objects centered in the eyepiece.

A set of exercises could be performed; for example, the student could be asked to locate a particular star without computer aid. Upon request, the system could indicate whether the correct star has been centered in the eyepiece.

This type of exercise would provide a smooth transition from indoor experience to practical telescope operation. Training drills may be also set up in the computer to teach various types of observations.

4.3 SUMMARY

The identification of numerous applications in the two areas of observing aids and teaching aids suggested that the TGDS would indeed be worth developing. It also provided the basis for the decision on which functions to implement as part of this thesis, as discussed in the following section.

5. PROTOTYPE SYSTEM OVERVIEW

In this section, specifications for the prototype system are laid out. An overview is then presented of a prototype TGDS designed to meet those requirements. The overview includes the major functional blocks, and operation in terms of the user interface.

5.1 PROTOTYPE SPECIFICATIONS

For the purposes of this thesis, it was decided not to attempt to implement all of the functions necessary for the applications described in Section 4. The prototype was intended to prove the feasibility of the system by implementing the most important functions.

It was decided to implement a system which could display star field graphics, telescope position, and object identification information in the eyepiece. Of the suggested applications, such a basic system could be used for locating objects, identifying objects, and for all three of the proposed training modes. The major system functions required are as follows:

- a) Real-time retrieval and display of star map data to the faintest magnitude possible.
- b) Display of information both in the telescope eyepiece and on the main computer display.
- c) Display of data on objects in the telescope field, including their designation and celestial coordinates.
- d) Automatic, accurate control of the telescope position.
- e) Automatic location of objects, given their name or designation.

A design objective was to make the system expandable so that the additional functions for other applications could be easily added.

In order to evaluate the performance of the system, a set of design criteria is required. This includes criteria for response time of both the graphics screen and telescope motors, and the accuracy of the video display and motors.

Using the commercial motor controller supplied with the telescope, the maximum slew speed is four degrees per minute; this requires over 10 minutes to slew 45 degrees. A dramatic improvement is necessary for the system to be at all practical. Preliminary tests on the motors indicated a factor of five improvement was possible, reducing the time for a 45 degree slew to two minutes. In comparison, an experienced observer will require about five minutes to manually locate a faint object.

The response time of the graphics display is also important. This is very important for operations where a fast response is required, such as scrolling the star images on the screen. A response longer than one to two seconds would be very annoying to the user. More complex operations such as searching the database for an object should not take more than about 10 seconds.

In terms of pointing accuracy, the telescope motors are capable of moving the telescope 3.125 arc seconds in a half-step mode. At this accuracy, some image vibration due to normal tracking is noticeable at high magnification; any further loss of precision would compromise the image quality of the telescope.

In order for the image of the stars and the graphics in the telescope eyepiece to match, the graphics display must have good linearity, and the motor control must be accurate. If the linearity error is less than 1% of the field size, it will not be very noticeable. Over small distances, the motor drive should also be accurate to 1% of the field size.

Over larger distances, it is extremely difficult to achieve very high pointing accuracy. The effects of polar alignment errors, mounting flexure, and atmospheric refraction become worse with increasing distance. Even with a 10% error, however, the

correspondence of the graphics image and the star images should still be obvious. For this reason, a 10% error shall be deemed acceptable for the prototype. If the errors are this large, it will be necessary for the realignment procedure to be very fast and simple.

To summarize, the following performance goals must be realized:

- a) The maximum slewing speed for the telescope motors shall be at least 0.4 degrees per second.
- b) Response time for operations requiring nearly instantaneous response, such as scrolling the display, shall be less than two seconds.
- c) Response time for more complex operations, such as database searches, shall be less than ten seconds.
- d) The telescope shall be capable of being moved in 3.125 arc second increments.
- e) The linearity of the eyepiece graphics screen shall be within +/- 1%, measured as a fraction of the screen size.
- f) Over slewing distances of less than three eyepiece field diameters, the motors shall be accurate to within +/- 1% of the field diameter.
- g) Over slewing distances of 45 degrees or less, the motors shall be accurate to within 10% of the field diameter.

The accuracy for very large slewing distances was not specified since it is faster to first manually realign the system on a star in the vicinity of the destination. This is necessary in any case if the telescope crosses the meridian, due to the design of the telescope mounting used.

The system designed to meet these requirements is described in the following sections.

5.2 SYSTEM HARDWARE

The basic elements of the system are shown in Fig. 5.2-1. The system consists of a telescope with motors on both axes, an optical assembly for projecting a graphics image into the eyepiece, a power supply for the optical assembly, a microcomputer, and a motor controller. As of 1987, there are several commercially-available high-end telescopes available with the required motors as optional accessories.

5.2.1 Telescope

A "Super-Polaris" Celestron-8 telescope was selected for use with the prototype (see Plate 5.2.1-1). The primary selection criterion was the availability of the instrument, which in fact made the realization of the prototype system possible. The telescope includes an optional stepper-motor drive.

Although shaft-encoders are the preferred method for sensing the telescope position, stepper motors can be used to provide an open-loop indication. The accuracy of such a system may be limited by step slippage and gearbox backlash. Further limitations of the particular instrument used includes a fairly low slewing rate, backlash problems in the motor drives, and some mounting flexure. The first two problems were compensated for in the design of the motor controller, as described in sections 6.2 and 8.

The telescope is equatorially mounted, as described in section 2.1. This is done so that the telescope can be rotated slowly to counteract the Earth's motion. This also makes the measurement of the celestial coordinates of an object much easier.

Fig. 5.2-1 Major System Components

**National Library
of Canada**

Canadian Theses Service

**Bibliothèque nationale
du Canada**

Service des thèses canadiennes

NOTICE

**THE QUALITY OF THIS MICROFICHE
IS HEAVILY DEPENDENT UPON THE
QUALITY OF THE THESIS SUBMITTED
FOR MICROFILMING.**

**UNFORTUNATELY THE COLOURED
ILLUSTRATIONS OF THIS THESIS
CAN ONLY YIELD DIFFERENT TONES
OF GREY.**

AVIS

**LA QUALITE DE CETTE MICROFICHE
DEPEND GRANDEMENT DE LA QUALITE DE LA
THESE SOUMISE AU MICROFILMAGE.**

**MALHEUREUSEMENT, LES DIFFERENTES
ILLUSTRATIONS EN COULEURS DE CETTE
THESE NE PEUVENT DONNER QUE DES
TEINTES DE GRIS.**

Plate 5.2.1-1 Celestron-8 Telescope

The optical system of the telescope is a Schmidt-Cassegrain. In such a telescope, the light passes through the optical system as shown in Fig. 5.2.1-1. The diameter of the optics is 200 mm, which is quite reasonable for serious astronomical work. In this type of telescope, focussing is achieved by moving the primary mirror instead of the eyepiece. Since it is not necessary for the viewing apparatus to move, a very stable support is possible for the graphics display assembly.

5.2.2 Optical Assembly

The role of the optical assembly is to superimpose a graphics image on the telescope image. A small off-the shelf commercial CRT was selected since it could provide reasonably high-resolution graphics images. In a commercial version of the TGDS, a higher-resolution CRT – perhaps specially designed – would provide superior results. A partially-aluminized mirror was used to combine the light from the telescope with the light from the CRT.

The construction of the optical assembly is described in more detail in section 6.1. A photograph of the completed unit is shown in Plate 5.2.2-1.

5.2.3 Computer

An IBM-PC compatible computer was selected since a wide variety of graphics cards are available for such systems (see Plate 5.2.3-1). An ATI "EGA Wonder" card was selected for the graphics controller, for reasons described in detail in section 6.3.

The computer's EIA standard RS-232 serial interface is used to communicate with the stepper-motor controller. This permits the computer to control the telescope's position in real time.

The computer also includes a large screen video display, keyboard, 8087 math co-processor, and a 20 megabyte hard disk.

Fig. 5.2.1-1 Telescope Optical System

Plate 5.2.2-1 Optical Assembly

Plate 5.2.3-1 Main Computer

5.2.4 Motor Controller

A small 6809-based micro-controller was built to control the telescope motors (see Plate 5.2.4-1). The motor controller replaces a commercial model which was inadequate for the TGDS requirements. The motor control design was intended to:

- a) achieve higher slewing speeds
- b) reduce step slippage by increasing torque
- c) provide a compatible interface to the main computer
- d) relieve the main computer of the processing load
- e) ensure that the timing of the signals to the motors is accurate (delays due to disk accesses on the main computer preclude direct control).

Improved performance over the commercial model was achieved by using several techniques: (a) the slewing rate was increased by using a very high overdrive voltage pulse at the start of each step, (b) the software was designed to eliminate gear train backlash by always approaching the desired location from the same direction, and (c) step slippage was greatly reduced by ramping the motors up and down gradually, and by changing from half-step to full-step to overdrive pulse mode as the speed increases.

Plate 5.2.4-1 Motor Controller

5.3 OPERATION AND USER INTERFACE

A good telescope user interface is the main goal of the TGDS system. The design of the video display and keyboard interface was iterated several times to obtain a good interface. The characteristics of the interface is described in the following sections.

5.3.1 Display

The major design consideration was reading the display under low light conditions. Since the image in the eyepiece is fairly small, and the large screen may be viewed from a fair distance, the lettering was made very large. To avoid interfering with night vision, all textual information was displayed at moderate intensity. To eliminate the need for switching screen modes frequently, all text was displayed simultaneously with the graphical sky image using bit-mapped graphics. The exact same image is displayed on the main screen and in the eyepiece at all times.

Upon starting the program, both the display presents a view of the sky directly overhead. Once the system has been aligned with the telescope, the display shows the stars in the direction the telescope is pointing. Five scales are available, ranging from a wide-field view of approximately 50 degrees, to a view corresponding to the telescope field of approximately 0.35 degrees.

An auxiliary display mode shows the stars above the horizon in any direction. This is useful for quickly determining which objects are visible at any time.

5.3.2 Keyboard Control

The main consideration in designing the user interface was to provide an extremely simple, fast way to control the system. Several approaches were tried, including selecting commands from a menu using a small hand paddle. A function key approach was the most successful since access is much faster once the key locations have been memorized. The major disadvantage of this is that the keyboard is fairly large, and therefore must be placed on a small table next to the eyepiece. Since the operator rarely

needs to touch the manual telescope controls it is also possible to carry the keyboard in one hand.

An advantage of the function key approach is that a small hand-held keypad can be added to the system. This keypad would supplement the keyboard, providing most of the function keys required to operate the TGDS. This also provides a familiar user interface for most observers, since telescopes are often controlled with a small hand paddle. Since the prototype was intended as a proof-of-concept system, such a keypad was not provided.

The functions of the various keys used in the TGDS prototype system are described below. The numerical keypad and function keypad on the IBM PC compatible keyboard were used.

The numeric keypad is used to control the telescope and star display position. By pressing a key, the position can be moved by an increment in any of eight directions corresponding to the points on a compass. The "8" key, for example, moves the telescope and screen position north. The size of the increment is controlled by pressing one of five keys on the function keypad. These keys are arranged in a row from smallest to largest increment for convenience.

The scale of the display is controlled with five function keys, to select 50, 30, 5, 2 and 0.35 degree views. These keys are assigned to the same five keys as the increment control, and the "CTRL" key is used to distinguish which operation is performed. The 0.35 degree scale is the field of view of the telescope using a 40 mm focal length Plossl eyepiece with the display unit attached. For this reason, viewing at this scale will be referred to as **telescope view mode**.

The telescope does not move unless the display is in telescope view mode. This permits the user to move to the large scale display, center an object, and then return to the telescope view mode. At that time, the telescope automatically moves to the new position. If the observer wishes to move the telescope without entering the telescope view mode, pressing the space bar starts the telescope moving.

The remaining functions of the TGDS are accessed through the remaining five keys on the function keypad, in conjunction with the "CTRL" key. The program can be easily modified to assign the various functions to any key on the keyboard. Each of the functions are described below.

Two function keys control the information displayed. Pressing one key toggles the star display on and off. This allows undisturbed views of the actual sky. The second key allows the user to switch on a display showing the present telescope position; a small cross-hair is also displayed at the center to indicate the exact location. Either of these two displays can be shown simultaneously or independently.

Functions were provided for locating and identifying objects. The first key, when pressed, displays the identity of an object centered in the cross-hairs. When the second key is pressed, the computer prompts the user for the name of an object. If the named object is within or just beyond the present display, it is be automatically centered on the screen and labelled. The constraint that the object be nearby was imposed to avoid a time-consuming scan of an extremely large database.

One command was provided which allows the user to type in a new position directly. This could be used if, for example, the observer wishes to slew the telescope to the published location of a comet.

One key was provided to allow adjustment of the telescope to match the display. When the key is pressed, an indicator is illuminated on the screen. Pressing the key once permits the telescope to be adjusted without moving the screen. Pressing the key again permits the screen to be adjusted without moving the telescope. Pressing the key a third time returns the system to normal operation. As an aid to aligning the instrument, all stars appear as small crosshairs when adjusting the instrument in the telescope view mode.

Another key allows the user to make adjustments to the scale of the image in two dimensions. This allowed adjustment for differences in optics, focus, or CRT adjustment. Yet another key permits the user to change the default viewing scale of the telescope view mode.

Keys were also provided to command a re-calculation of the displayed image, and to exit the display mode to a system status menu. The status menu permits the user to change or view standard and sidereal time, determine which constellations are above the horizons, and to exit the program.

6. **HARDWARE DESIGN**

6.1 OPTICAL ASSEMBLY

The optical unit was designed to superimpose graphical information on the image of celestial objects in the telescope eyepiece. Construction of the unit required consideration of both mechanical and optical design. These two aspects of the design are discussed in the following sections.

6.1.1 Mechanical Design

The optical assembly is shown in Fig. 6.1.1-1. The CRT unit selected includes a 2.54 cm screen complete with NTSC-compatible drive electronics powered from 5 volts. The unit is normally used as the viewfinder on a home video camera, and thus is readily available.

The CRT is mounted inside an assembly which has a rotatable mirror at the end, allowing adjustment of the image left-to-right. The entire module may be moved slightly to provide adjustment in the perpendicular direction.

A second mirror reflects the CRT image towards the relay lens, diaphragm, and partially-aluminized mirror. The relay lens focuses an image of the CRT at the focal plane of the eyepiece.

The light from the two sources could have been combined using one of several techniques, such as using a thin glass plate mounted at 45 degrees to the light path of the telescope. The plate would not have anti-reflection coatings on one side, so that the light from the CRT could be bounced off one surface towards the eyepiece. This technique was not selected since the plate would have to be custom-manufactured. Instead, an off-the-shelf partially-aluminized mirror was selected. The partially-aluminized mirror

Fig. 6.1.1-1 Optical Assembly Design

allows 20% of the light from the CRT to reach the eyepiece, while still reflecting 80% of the light from the telescope towards the eyepiece.

A production version of the optical assembly would likely contain a lever to remove the image combiner from the telescope light path when high-quality viewing is required. If a partially-aluminized mirror is used for the combiner, the reflectivity could be also increased to reduce image degradation when in the light path.

The optical components are epoxied to specially-made aluminum brackets. Some degree of adjustment was provided by making the mounting holes slightly larger than required.

An eyepiece holder on the end of the unit holds a "star diagonal" which reflects the light 90 degrees. The star diagonal in turn holds the eyepiece. This produces non-inverted images, and places the eyepiece in a more accessible location.

The assembly attaches to the telescope as shown in Plate 6.1.1-1. The orientation of the unit relative to the telescope may be adjusted by rotating the entire unit until the star images coincide with the graphics. A production model would include a more precise adjustment which would rotate the CRT.

The optical assembly also contains a potentiometer for adjusting the display intensity, and a 5 volt power regulator. Power is supplied by an external 12 volt supply in order to keep the unit lightweight. The power cord also carries the video signal from the computer.

Plate 6.1.1-1 Optical Assembly and Telescope

6.1.2 Optical Design

This section describes in detail the design of the optical system outlined above. The most important parameter of the optical assembly is the scaling of the CRT image relative to the image of the sky. A detailed description of the necessary calculations follows.

The size of the CRT image is determined by the focal length of the relay lens and eyepiece, and their relative placement. Due to the dimensions of the box required to fit the CRT, and the assembly used to hold the eyepiece, the distance between the eyepiece focus and the CRT was set to 43 cm (approximately, since some adjustment range was provided). The diagonal size of the CRT is 2.54 cm, so the smallest dimension is approximately 1.5 cm for an aspect ratio of 4:5. The 40 mm Plossl eyepiece has an apparent field of view of 43 degrees, corresponding to an image at the focal plane of the eyepiece of 3.0 cm. The following equation can be used to determine the position of the relay lens [9]:

$$\frac{I}{O} = \frac{q}{p} = \text{Magnification}$$

Where:

- I = Size of Image
- O = Size of Object
- q = Distance Between Relay Lens and Eyepiece
- p = Distance Between Relay Lens and CRT

We require an image size at the focal plane of the eyepiece of 3.0 cm, and the actual CRT is 1.5 cm. Therefore, a magnification of $3.0 / 1.5 = 2.0X$ is required. Since the total distance is 43 cm, this requires q to be approximately 29 cm, and p to be approximately 14 cm. A range of adjustment was provided to adjust the size of the CRT image.

The field of view of the telescope/eyepiece combination is calculated from the focal lengths of the telescope and eyepiece as follows [9]:

$$M = FL_T / FL_E$$

$$AF = EF / M$$

Where:

FL_T = Focal Length of the Telescope
 FL_E = Focal Length of the Eyepiece
 M = Magnification
 EF = Apparent Field of Eyepiece
 AF = Actual Field

The Celestron-8 telescope has a focal length of 2000 mm, and the eyepiece has a focal length of 40 mm, giving a magnification of 50X. Since the apparent field of the eyepiece is 43 degrees, the actual field of view is 0.85 degrees.

This calculation is extremely inaccurate if applied blindly using the telescope specifications, however, since the focal length of a Schmidt-Cassegrain telescope changes with focus. The reason for this is that the focussing mechanism moves the primary mirror closer to the secondary. The effect is quite large, so a small perturbation can have a fairly large effect. A quick comparison of several Celestron-8 telescopes revealed large production tolerances, so it was decided to measure the field size empirically.

The final determination of the appropriate scaling factor was made by running the TGDS and superimposing a graphics image of several stars over their actual telescope image. By adjusting the graphics display for a good match, the field size was determined to be 0.35 degrees. This is much smaller than the previously calculated 0.85 degrees due to the large focal length change.

In order to ensure that a good image of the CRT and telescope field was produced, two steps were taken. First, all surfaces inside the optical assembly were painted flat black. Second, the diaphragm was used to reduce spherical aberration. Spherical aberration is caused by the use of spherical surfaces on simple lens elements.

Parallel beams of light at the edge of the lens and the center do not focus at the same point. The diaphragm reduces the distortion by covering up the outer edges of the lens. This has the added advantage of reducing the intensity of the CRT image to a level suitable for the dark-adapted eye.

Finally, it should be noted that it is necessary to accurately focus the image of the CRT and of the sky simultaneously. Moving the eyepiece in and out of the holder changes the focus of both the telescope image and the CRT. Adjusting the telescope only affects the telescope image. Therefore the CRT image must be focused first by adjusting the eyepiece, then the telescope may be adjusted.

6.2 MOTOR CONTROLLER

The motor controller was designed using a microprocessor instead of discrete hardware due to the complexity of the control tasks required. A 6809 microprocessor was selected due to its low cost, and because its instruction set is probably the most powerful of any 8-bit processor. This made the software design task easier.

The internal construction of the motor controller is shown in Plate 6.2-1. The controller includes a power supply, microcomputer, and driver electronics.

The power supply is shown in Fig. 6.2-1. It supplies +5, +10, -10, and +48 volts. Only the 5 volt supply is regulated.

The microcomputer is shown in Fig. 6.2-2. It includes a 6809 microprocessor, EPROM, RAM, parallel interface, serial interface, and timer. The serial interface provides communications with the main computer. The timer produces the baud rate for the serial interface, keeps track of sidereal time, and determines the stepping rate of the motors.

The motor drivers are shown in Fig. 6.2-3. Since the telescope has bipolar stepper motors, only four drivers are required per motor. Each driver is controlled directly by the parallel port. Two additional drivers allow the microcomputer to boost the drive voltage to 48 volts for a brief period of time. This increases the rate of change of the current in the windings, thus increasing the motor torque [10]. The overvoltage drivers have an RC time constant in the circuit which limits the duration of the overvoltage pulse. This prevents overheating or damage in the event of a software error.

Spare pins on the parallel port are connected to an extra connector. This allows the addition of a hand paddle for more convenient control of the telescope.

Plate 6.2-1 Motor Controller Electronics

Fig. 6.2-1 Motor Controller Power Supply

Fig. 6.2-2 Motor Controller Microcomputer

Fig. 6.2-3 Stepper Motor Drivers

6.3 VIDEO CONTROLLER

In the case of the video display controller, the problem is not one of designing hardware, but rather of selecting the off-the-shelf hardware required to produce a cost-effective solution for the prototype.

Until recently, displays on IBM-compatible computers provided image differentiation primarily by colour and not by grey scale. For example, the IBM EGA standard can display 16 colours simultaneously out of a palette of 64, insufficient to provide a grey scale display on a colour EGA monitor. The ATI "EGA Wonder" and the Techmar "Graphics Master" are almost unique among inexpensive, PC-compatible display cards in providing 16 grey levels at a resolution of 640 x 200 lines and above.

Since the original IBM standards, CGA and MGA, used different electrical video interfaces, the ATI "EGA Wonder" card was designed to accommodate both types of monitors (ie. analog composite and digital RGB) in any video mode. This display card was selected since it could produce 640 x 350 pixels with 16 approximately linear grey levels on a standard NTSC-compatible "composite" monitor, such as the off-the-shelf CRT unit used in the TGDS optical assembly.*

The recent advent of the IBM VGA analog display standard, which can display 16 colours out of a palette of 256K at either 640 x 480 or 640 x 350, means that a future TGDS with a colour VGA monitor could provide colour as well as brightness cues in star-field images.

* In late August, 1987, Princeton Graphics, New Jersey, announced the first monochrome EGA display (the MAX-15) capable of simultaneously showing 16 shades of grey out of a palette of 64 linear, accurate shades of grey. A miniature version of this, driven from any EGA card, would produce superior results to the system used here.

7. MAIN COMPUTER SOFTWARE DESIGN

The main computer software was written and executed on an IBM-PC using a Turbo Pascal compiler. This permitted rapid development of fast structured code. The Turbo Pascal version used utilizes the 8087 math co-processor for maximum floating-point processing speed. Complete listings of the software have been included in Appendix D.

There are three main software components: **database management, graphics generation, and telescope control.** Each of these are described in the following sections. A brief description of the program structure follows.

7.1 DATABASE MANAGEMENT

In order to provide graphics of star images to a sufficient level of detail, a number of databases were used. These include the following:

- A public-domain database of the brightest stars, called "STARS", containing about 1700 stars;
- The Yale Bright Star Catalog (YBSC), listing approximately 4000 naked-eye visible stars;
- The Smithsonian Astrophysical Observatory (SAO) catalog of 259,000 stars to 9th magnitude.

The program was designed to accept additional databases with little modification.

There were several reasons for the choice of three databases. First, it was desirable to limit the number of stars in memory at any one time to about 1000, in order to assure real-time response. Thus, only the brighter stars were displayed on the wide-field views. Second, in order to cover stars to the fainter magnitudes visible in telescopes,

the SAO catalog was essential. The SAO catalog does not, however, include the brighter stars. Thus, the YBSC catalog was required as well.

7.1.1 Database Storage

The databases, as acquired for the project, were set up in a variety of poorly-documented formats. The first task was therefore to decipher the databases and reformat them into a consistent form.

Since a large volume of data was required, there are some problems in accessing it fairly quickly. Several possible techniques were considered, including storing a large number of roughly square overlapping map segments. With any reasonable overlap, however, the volume of data grows rapidly. The data pre-processing could include the map projections, but a variety of map sets would be required for different scale displays, and calculating the actual star coordinates from this data might be difficult. This technique was rejected due to the amount disk space required.

A more obvious solution was to sort the data into a set of fixed-size files, ordered by their celestial coordinates. With a lookup table to determine the correct files to read given a particular coordinate range, the data could be located very quickly. In order to assure coverage of the entire screen, it might be necessary to read in two or more files in sequence, which would slow down the processing. This solution was adopted however since it only required a reasonable amount of disk space.

For speed purposes, the databases were not packed in any way. The data for each star was stored as two 64-bit real numbers for the coordinates, a 16-bit integer representing the star's magnitude times 100, and a six-character name. As a result, the three databases consumed approximately 10 megabytes of disk space.

The three databases were maintained separately. The "STARS" database was kept as one file; the "YBSC" database was broken into four files; and the "SAO" database was broken into 101 files. Each individual file could be read in one operation into a 64 kilobyte memory array.

7.1.2 Pre-Processing

The databases were extensively pre-processed to facilitate rapid access. First, they were translated from various packed formats into homogeneous, directly readable files.

The databases were sorted into increasing order in Right Ascension. They were then broken up into equal-sized files, and each individual file was then sorted into increasing order in Declination. A table indicating the Right Ascension range for each file was then generated and stored for access by the display program.

Thus, when the display program required stars for a certain area, it could check the table to determine which range of files to read. It could then scan through those files for the range of declination required, and process only the stars of interest. Since each file was sorted by declination, the correct range of data within the file could be located very quickly.

An additional step was required, however, due to the precession of the earth's axis. The earth's axis wobbles slowly in a complex fashion due to a variety of effects. The result of this is that the celestial coordinate system is not an inertial frame: the Right Ascension and Declination of a star changes with time. This calculation is complex and time-consuming, and would slow down the display program significantly.

In order to provide the fastest display time possible, the coordinates for all stars in the databases were precessed in advance of the observing session using a utility program.

7.2 GRAPHICS GENERATION

Once the star data of interest has been obtained, an image must be generated. Several steps were required, including calculation of X-Y coordinates, intensity, and actual generation of the graphics.

7.2.1 Projection

The problem of representing the spherical celestial coordinates on a flat plane is that of a map projection. The same map projections are used for geographic maps as for star maps, the only difference being that east and west are reversed on the sky maps. This is because the "celestial sphere" is viewed from the inside.

The problem is then selecting which projections to use. Various types have been developed, each of which has different types of distortion. Since some projections perform well near the poles, others at the equator, etc., the program was designed to use several different projections. The decision on which to use depended on the declination range of the display.

Near the equator, a cylindrical projection was used, being quite accurate and easily calculated. Further north, the Lambert Conformal Conic projection was used. The parameters of the projection are adjusted depending on the position of the center of the map, and the scale. This minimizes the amount of distortion. Fortunately, the distortion problems are the least severe in the telescope view mode, which has the smallest scale.

Since map projection computations are quite time consuming, every effort was made to optimize the processing time. The calculations were coded carefully to minimize the number of floating-point operations required. For example, some of the calculations are identical for each star. This information is calculated once and stored for use during the projection loop.

7.2.2 Real-Time Display

One requirement of the system was that the user be able to slew the telescope a small distance, with the video image following the telescope image in real time. The requirement was for the screen to update within two seconds. Unfortunately, even if the correct range of data was in a single file, ready to read and display, it would take too long to load and display it. For this reason, the database could not be reloaded from disk each time the screen was scrolled.

A compromise was therefore made: a larger set of data than nominally required would be loaded from disk. The user could then scroll around within a limited area of the sky before running out of data. New data could be reloaded when required.

Even so, just performing the map projections each time the screen was scrolled was too slow. Therefore, it was decided to perform the map projections while the data was being loaded. The operation would then be reduced to calculating the required vertical and horizontal shift to the fixed data in memory. This would require only one map projection calculation to determine the offset from center. An area approximately six times larger than the screen is always loaded so that the view can be shifted a substantial distance before running out of data.

A new display can be generated in 1-2 seconds using this technique, although loading a new star field display and computing the map projection can take up to 10 seconds on an 8 MHz 8088-based PC with an 8087. For short slews, the telescope takes approximately 2 seconds to move, so the response time of the TGDS is quite suitable. More detailed measurements of the response time have been included in Section 9.3.

Another decision was how to decide when to load new data from the disk. Arbitrarily doing this when the user approaches the edge of the data would cause the telescope and user interface to suddenly halt for ten seconds; this would be extremely annoying. It was therefore decided to let the user request new data when required.

Using a faster computer could reduce the time required for loading and processing the data enough so that the reloading operation would not take very long. The

trigonometric calculations required for the map projection could be speeded up by a factor of 25 using a computer with an 80387 math co-processor, primarily due to a new sin/cos instruction. This co-processor would be fitted to an 80286 or 80386-based computer, which can process data much faster. The "PC/AT" computers also have much faster disk drive controllers. With this added processing power, it would become feasible to introduce automatic loading and re-calculation. The added cost, however, might not be justified.

The display itself was double-buffered. When the operator presses a slew key, a new graphics image is written out to the background buffer. The background and foreground buffers are then swapped. This makes the writing operation transparent.

One problem with the system was that a quick series of scroll requests from the user resulted in a very slow series of screen refreshes. This was a problem since the user normally requires several key operations to reach the desired destination. In order to eliminate the useless intermediate screen refresh operations, the computation routines were designed to automatically terminate whenever a key was pressed. When no more keys were being pressed, the calculations were re-started from the beginning, with the latest position and scale information. As a result, the processing delay was always 1-2 seconds after the last key was pressed. This same technique was applied to the disk loading operations, so that if the wrong scale key is pressed, the operator does not have to wait two or more 10 second intervals.

7.2.3 Screen Interface

The image of a star is easily represented by illuminating just one pixel on the screen. The magnitude of the star can then be indicated by varying the brightness of the pixel. The problem is that the range of brightness of the stars far exceeds the contrast range of a CRT.

For example, the wide field views only show the brightest stars. Since only 16 gray levels are available, the darker shades should not be reserved for stars which are not being displayed. Instead, the brightness is adjusted so that the brightest star in the sky is

displayed with the brightest shade. The faintest star displayed at that scale is then assigned the darkest shade.

Upon changing to a narrow field of view, however, very few bright stars are visible, and many stars fainter than previously shown are present. The brightness range is therefore adjusted to increase the brightness of all stars. Through experimentation, a three-level approach was found to work well, as it corresponded to the three levels of databases. This provides the most natural and realistic display at all scales: one range of scales represents naked eye viewing, one range represents binocular or finder-scope viewing, and one range represents telescope viewing.

Note that a common imperfection of CRTs – that brighter images "bloom" or grow in size slightly – does not significantly impair the image quality since it produces a result similar to photographs: scattering in photographic emulsions cause brighter stars to appear larger. Therefore the entire range of brightness available on the display were used.

The star image is displayed using special routines which calculate the correct bit patterns to output to the hardware. Look-up tables were used extensively to provide an extremely fast graphics driver. Routines were also written for displaying text while still in the graphics display mode.

It will be necessary in the future to display non-stellar objects such as galaxies, nebulae, etc. A special character font was therefore designed which includes standard symbols for these non-stellar objects.

7.3 TELESCOPE CONTROL

Control of the telescope is performed through the computers' RS-232 standard serial interface. The main computer calculates the number of steps needed to move the telescope the required distance. This information is sent to the motor controller using a simple protocol, as demonstrated by the following examples:

<u>Command</u>	<u>Action</u>
R0100	Move 100 (hex) steps in increasing Right Ascension
DFF00	Move 100 (hex) steps in decreasing Declination (FF00 = -0100 in two's complement)
M01000FFF00	Slew 1000 (hex) steps in increasing RA, and 100 (hex) steps in decreasing Declination
S	Stop immediately

The number of steps actually refers to the number of half-steps taken by the motors. The protocol was designed for two modes of operation. One is used for short distance slews, such as for centering an object in the eyepiece. The other mode is used for long distance, high speed slewing.

The "R" and "D" commands allows the user to move the telescope a small distance at a fairly low speed. Since the user can press a slew key several times in succession, the motor controller accumulates all the commands.

The "M" command allows the user to slew the telescope a long distance. The distance in both directions is specified in the command, and larger numbers are permitted.

Only one slew command is permitted at one time. This is natural, however, due to the design of the user interface. The observer finds the object on a large-scale view mode, then switches to the telescope view mode. At that time, the computer transmits the slew command. While moving around in the telescope view mode, the short slew commands are used.

7.4 PROGRAM DESCRIPTION

A simplified structure chart for the program is shown in Figure 7.4-1. This diagram shows only the most important modules for clarity.

The structure of the program is primarily driven by the user interface. The mainline initializes the program, and calls the `Main_Menu` routine. This routine provides an auxiliary menu for accessing testing modes, setting date and time, and displaying horizon views. Upon initialization, `Main_Menu` passes control directly to `Generate_Graphics` without user input.

`Generate_Graphics` decides on the test modes, which enable plotting of grids and manually entered coordinates for testing map projections. Normally, control is passed on the `Draw_Stars`, which is the main control routine for the program.

`Draw_Stars` is responsible for keeping the star map graphics updated, and displaying text on the graphics screen. The majority of the user interface functions are handled by `Process_Key`, which is called whenever a key is pressed. In addition to initiating many functions directly, `Process_Key` sets some flags to indicate to `Draw_Stars` when some operations must be performed.

`Process_Key` was added as a subroutine of `Draw_Stars` when the program was switched from the original menu-driven approach to a more convenient function key approach. An alternative would have been to make `Draw_Stars` a subroutine of `Process_Key`. The choice is largely irrelevant because the two routines behave as co-routines, using flags to pass information between them. The flag-passing mechanism maintains a record of interrupted operations. If, for example, a database reload operation is interrupted by a function key action, the flag requesting reload is not cleared. When control returns to `Draw_Stars`, it will start the reload operation again.

Fig. 7.4-1 Main Program Structure Chart

Process_Key controls the position of the telescope directly, using the **Move_Scope** routine. While in the telescope view mode, **Move_Scope** sends a command to the motor controller to move the telescope slowly over short distances. While in other modes, **Move_Scope** records the movement but takes no action. Whenever the space bar is pressed, or telescope view mode is entered, **Slew_Scope** is called. **Slew_Scope** determines whether the telescope needs to be moved, and sends a high speed slew command to the motor controller if required.

Process_Key also updates locations indicating the present display scale, display position, slewing increment, and display mode. It also calls **Identify_Object** and **Locate_Object** when requested. Whenever these operations require refreshing the screen or reloading the database, **Process_Key** sets the appropriate flag.

When requested (by **Process_Key**) to reload the data from disk, **Draw_Stars** calls **Pre_Calculate**, which calculates new position and scale-dependent parameters for the map projection. It then calls **Read_File**, which locates new data on the disk. **Read_File** calculates the range of Right Ascension and Declination required, determines which database files are necessary, and calls **Load_Buffers** to get the data. For each file, **Load_Buffers** calls **Load_File** to read a file into a buffer, **Scan** to search for the beginning and end of the correct Declination range, and **Copy_Stars** to move the required data into the display buffer. While moving the data, **Copy_Stars** excludes stars in the file which are outside the desired Right Ascension range, and calls **Altazimuth** and **Project_Point** as necessary to perform the map projections. For the first-level database, which has only one file, **Read_File** skips **Load_Buffers** and calls **Load_File** and **Copy_Stars** directly.

When a flag is set indicating that the data must be re-displayed, **Draw_Stars** calls **Display_Stars**, which scans through the star data memory buffers. For each star, **Display_Stars** calls **Display** to output the data to the screen. **Display** is part of a file which is customized for the particular video display used; procedures were originally written for IBM-compatible CGA. The procedures were later replaced to with a version to drive a Techmar card, and finally with an EGA-compatible version. This provided the

flexibility required to develop the software before the video display decision had been made.

The remaining procedures will not be described in detail. These procedures perform map projections, rotate from equatorial to altazimuth coordinates, manipulate the graphics screen, obtain different types of textual data from the keyboard, obtain real-time clock information from the operating system, calculate Julian date, perform conversion to sidereal time, and convert between radian, degree, and hour formats.

8. MOTOR CONTROLLER SOFTWARE DESIGN

The motor controller software was written in 6809 assembly language. A cross-assembler was used on an IBM-PC, and the code was transferred to the controller board via a serial interface. A bootstrap EPROM on the controller board loaded the software into memory and executed it. A complete listing of the motor controller software has been included in Appendix D.

The following sections describe the major functions of the motor controller software. A description of the program operation follows.

8.1 MOTOR CONTROL

Using stepper motors for the telescope drive introduces several problems. For the highest accuracy, slippage must be avoided. The step size must be fairly small, in order to avoid degrading the image with vibration. At the same time, however, the motors must be able to slew the telescope at a fairly high speed.

In order to address these constraints, it was decided to operate the motors in three modes: half-step, full-step, and overdriven full-step. The half-step mode was used for tracking and short slews to provide the smallest, smoothest steps. The full-step and overdriven full-step modes were used to obtain higher torque. Also, the motors were ramped up and down in speed to prevent slippage when starting and stopping.

In the resulting design, when a high-speed slew is requested, the motors are gradually accelerated. As the speed increases, the motors are switched to full-steps. This is done at a phase of the motor where changing modes does not affect the winding currents, thus providing a smooth transition. As the speed increases still further, a 48-volt overdrive pulse is applied at the start of each step, to increase the torque. Changing between the different step modes was simplified by using a set of three lookup tables to determine the correct output.

8.2 TIMERS

The motor controller unit contains a 6840 timer chip, which has three timers. One timer is used to generate the baud rate for the serial interface. The second timer is used to drive a sidereal real-time clock. The third counter was used for timing of the step rate. The latter two timers generate interrupts.

8.2.1 Sidereal Timer

The real-time clock timer was carefully set up to provide a rate accurate to within four seconds a day. This was important since the Right Ascension axis of the telescope must be turned at a constant rate to track the motion of the stars.

The clock was set up for sidereal time since this matches the rotation of the earth relative to the stars. A sidereal day is approximately four minutes shorter than a solar day: the earth actually rotates 366.25 times a year, not 365.25. The one day difference is due to the revolution of the earth around the sun.

In order to track the stars accurately, the telescope must make one revolution in one sidereal day. At 414,720 half-steps per telescope revolution, the half-steps must be performed every 0.2077645 ms (see calculations in Appendices A.1.4 and A.1.5).

In the resulting design, interrupts actually occur at four times the nominal rate due to the limitations of the 6840 timer. The interrupt routine therefore requests a single half-step every fourth interrupt.

8.2.2 Motor Timer

The rotational speed of stepper motors is controlled by changing the winding currents at various rates. Since two motors were used, an obvious technique would be to use two separate timers. The timers would be initialized to the required duration between steps, and each motor could be run at different speeds. Unfortunately, the 6840 integrated circuit only has three timers available, and two of them were used for the sidereal timer

and baud rate generator. There was no room on the circuit card for a second timer circuit, so an alternative approach had to be taken.

One possible solution would be to run both motors at the same speed. On a typical slew, one motor usually runs longer than the other. Therefore, both motors would have to be ramped down to stop one of them, and then the other motor would have to be ramped back up again.

This difficulty was overcome by running the timer at a fixed high rate. The duration between steps was then determined in software by counting out the desired number of interrupts. In this way, one counter could be used for each motor.

The problem with this approach is that it reduced the number of processing cycles between the interrupts. As a result, erratic timing resulted when both motors were running. The solution was to slow down the interrupt rate, and count fewer interrupts before stepping. This resulted in slightly more coarse speed control, but eliminated the timing problems. An alternative solution would have been to use a faster version of the microprocessor and interface circuits. Since the motors performed well even with the coarser speed control, it was decided not to use the latter, more expensive approach.

8.3 BACKLASH CONTROL

A major problem with the telescope mounting was the very bad backlash in the motor gearing. This could have been corrected at a large expense by replacing the telescope mounting. Instead, it was decided to have the motor controller run out the backlash.

The motor controller was designed to always approach the requested position from the same direction. The method was to back off 32 steps in a particular direction, and then reverse the motors again for 32 steps.

When moving at low speed, this was only done when the motor direction was reversed. After slewing at high speeds, however, the backlash correction was always performed in case inertia carried the telescope past the desired point.

8.4 PROGRAM DESCRIPTION

The structure of the motor controller software is shown in Figure 8.4-1. Each of the major modules will be described in detail.

The Step routine outputs the necessary bit patterns to the stepper motor I/O ports. The routine is table driven, and uses a different table for half-step, full-step, and overdriven full-step modes. Input data tells the routine to step "up" or "down", which table to use, and which motor is being stepped.

The main Interrupt Service Routine (ISR) is interrupted by the 6840 timer chip at two different rates. One rate is used for the motor timing ISR, and the other for the sidereal time ISR. The main ISR clears the interrupt, determines which type of interrupt was requested, and calls the appropriate routine.

The motor timing ISR is quite complex. The routine alternates between the two motors on successive interrupts, and maintains a complete set of status memory locations for each. For clarity, the two sets will not be referred to explicitly in the following description.

The motor timing ISR uses one location to count interrupts. When the counter reaches a specified value, the motor is stepped. The terminal count value determines the stepping speed.

A second counter is used to indicate the distance remaining to be travelled. There are two separate "distance-to-go" counters; one for use on high speed slews, and one for low speed tracking. The high speed counter is more coarse, requiring 16 motor steps before being decremented. The 16 step count is maintained by a "sub-counter". This technique eliminates the need for multiple-precision arithmetic.

Fig. 8.4-1 Motor Controller Structure Chart

A state flag indicates whether high-speed or low-speed slewing is in effect. The flag is set to "Slew Request" by the main program. The motor timing ISR checks the flag, and switches it to "Slewing" only if the motor is at an even step (the motors are always moving at low speed when a slew is requested, so this cannot deadlock). In the low speed mode, the motor speed is fixed. In the slewing mode, however, the speed is adjusted dynamically.

To determine the speed required in the slewing mode, the ISR maintains a distance-travelled counter as well as the distance-to-go counter. The smaller of the two is used to decide which speed range to use. Thus, the motor speed ramps up when the slew is started, and ramps down when approaching the end.

Next, the motor is stepped in the appropriate direction by calling the Step routine. The direction is determined from the sign of the distance-to-go counter (in two's complement). The distance-to-go and distance travelled counters are then updated.

When the motor has been stepped, the routine checks whether the distance-to-go counter has been reduced to zero. If so, and a high speed slew is in progress, then the slewing state flag is cleared if both motors are at zero. If high speed slewing is not in progress, the routine checks a backlash flag. This indicates whether a backlash correction has been requested by the main program. If so, the low speed distance-to-go counter is incremented by a fixed value. Interrupt processing is then terminated.

The sidereal time ISR counts interrupts until four have occurred. It then increments the Right Ascension low speed distance-to-go counter by one step. If the motors are in low speed mode, then this causes the Right Ascension motor to move one half-step. This ensures that the telescope tracks the stars as the Earth rotates. If, however, the motors are in high speed slew mode, the counter accumulates the steps until the motors are switched back to low speed mode. At that time, the Right Ascension motor will move to catch up with the Earth's rotation.

The main program initializes the ports, timers and variables, and then enables interrupts. It then enters a polling loop which monitors the serial port. Upon receiving a

character, the routine checks for a legal command character, and reads in any appropriate additional digits. This is done using the Read, ReadDigit, ReadByte, and ReadWord routines. When the command has been loaded and interpreted, the mainline temporarily disables interrupts to prevent memory contention. It then updates the appropriate distance-to-go counters to request the desired action. The interrupts are then re-enabled.

9. RESULTS

In this section, the results of the TGDS development are discussed. This includes photographs of the actual hardware constructed, photographs of the various displays generated, a description of the operation of the system during a typical observing session, and the results of tests on system performance.

9.1 PROTOTYPE SYSTEM OPERATION

The complete system is shown in Plate 9.1-1. The telescope has the optical unit mounted on it, and the motor controller and optical unit power supply are on the tripod shelf. The main computer is located on a small table within reach of an observer standing at the telescope. The keyboard can be picked up for use when looking into the eyepiece. The large screen can be read from anywhere near the telescope. Despite the complexity of the system, relatively few cables are required to connect the various devices. This is important since too many cables can be dangerous to the operator. Cables are also a major cause of equipment failures in cold-weather observing.

9.2 QUALITATIVE TESTS

Tests were performed on the system under actual observing conditions at the Indian River Observatory near Ottawa. The system was found to be very easy to use. By manipulating the screen using the keyboard, the telescope could be quickly and accurately pointed at different objects. In most cases, it was much faster to use the system to locate an object than to do it manually.

The displayed information was easily read due to its large size. The main screen could be read from several feet away, and the eyepiece display was very legible.

The star field displays were very good, especially on the main screen. The contrast ratio between the dim and bright stars made recognition of patterns very easy,

Plate 9.1-1 Complete TGDS System

since it represented the appearance of the sky very well. In the eyepiece display the graphical star images did not appear as sharp points, but the contrast between bright and dim stars was still good. The larger size of the graphical star images was useful in distinguishing them from the real stars when comparing the two images.

The telescope image was noticeably fainter due to the light loss at the partially-aluminized mirror. In order to judge image quality, several diffuse objects were examined through the assembly. A slight loss of contrast was noted, but the resolution of the image was not degraded noticeably. Clearly, an arrangement for switching the partially-aluminized mirror out of the optical path would be useful.

9.3 SYSTEM PERFORMANCE TESTS

A number of tests were performed on the TGDS in the lab, and under actual observing conditions. Tests were performed on slewing speed, response time, CRT linearity, and pointing accuracy. Pointing and linearity errors were measured by re-aligning the telescope with a measured number of 3.125 arc second steps.

The maximum slewing speed could be raised to 0.5 degrees per second without motor slippage, which exceeded the design goals. The maximum speed was determined by increasing the speed until slippage occurred, and then reducing it by 10% for a safety margin. The repeatability was checked by slewing repeatedly between two test targets. If a slip occurred, the motor would be expected to slip by four full steps, corresponding to about 25 arc seconds, or about 2% of the field size of 0.35 degrees. This type of error was noticed occasionally (less than 10% of the slews) on the Right Ascension axis only, presumably due to higher loads on the axis. This type of error can be eliminated completely through the use of optical shaft encoders.

Motor slippage was not observed during short, low speed slews, and therefore no measurable errors were found (since the measurements were performed with the motors).

The linearity of the graphics display was tested by displaying a grid of 16 "stars", and moving the telescope to line up a target on each one in sequence. The worst

measurement gave an error of approximately 16 arc seconds, or about 1.3% of the field size, slightly worse than the target of 1%. This type of error could be reduced by calibrating out the distortion in the display in software, or by using a specially designed CRT unit.

During tests at the observatory site, larger errors were observed on long slews between different parts of the sky. This was likely due to some mis-alignment of the telescope axis with the Earth's poles. With a carefully adjusted mounting, the error on a 45 degree slew was always less than one arc minute, corresponding to 5% of the field size. This size of error was easily removed using the "Adjust Screen" mode. Flexure in the telescope mounting was also a problem. If the telescope was jarred the polar alignment would suffer, decreasing the pointing accuracy. One advantage of the TGDS system is that it can aid in adjusting the polar alignment by indicating errors in the star positions.

The response time of the graphics display software was measured using a stopwatch. It was found that the system could display slightly over 500 stars per second. A program modification was then added to display the number of stars in the memory buffer. The number of stars loaded at 6 different positions was determined (1 and 5 hours in Right Ascension; +45, 0, and -45 degrees in declination), using three different scales. The results were 1200 ± 160 , 710 ± 280 , and 44 ± 16 for scales of 50, 5, and 2 degrees, respectively. Therefore, at 500 stars per second, the total execution times were 2.4 s, 1.4 s, and 0.1 s. Except for the largest-scale view, the times were within limits. The reason for the slow response on the large scale view was that the second-level database was being used; field tests indicated that greater detail was useful at that scale even with the slower response.

The time required to load the database from disk and process it was also measured (values measured include the display time). The response time was measured for a five degree field at 15 positions (1, 2, 3, 4, and 5 hours Right Ascension; +45, 0, and -45 degrees Declination). Since the database search routine uses a linear search in declination, the higher declinations should be slightly slower, as was found to be the

case. The response times were 8.9 ± 0.3 s, 7.1 ± 0.1 s, and 6.9 ± 0.2 s, for +45, 0 and -45 degrees, respectively. The response times were all within the 10 second design goal. Using a binary search algorithm could potentially reduce the time required by about 25% at +45 degrees. The loading time at 0 degrees was comparable to -45 degrees due to the simpler map projection used near the equator. It is likely possible to improve the performance of the database loading operation further using optimized database access techniques and some code optimization.

9.4 A TYPICAL OBSERVING SESSION

In order to best describe the operation of the prototype TGDS, a typical observing session will be described. The task is to locate the faint star SAO 76136 in the Pleiades star cluster, in order to observe an occultation of the star by an asteroid.

When the program is started, it shows a wide field view of the stars directly overhead. Plate 9.3-1, shows the display for November 22 at 10:00 pm. The constellation Orion is at the lower left. Since Betelgeuse is a well-known star in this prominent constellation, the observer identifies it immediately, and decides to use it to align the system (this only need be done once at the start of any session). The observer sets the system in "Adjust Screen" mode, which does not cause the telescope to slew, and adjusts the display to center the star, as shown in Plate 9.3-2. With the graphics image of Betelgeuse centered, the observer presses a key to switch to telescope view mode (Plate 9.3-3). The stars change to cross-hairs to aid in exactly centering the star.

At this point, the observer manually aims the telescope at Betelgeuse, switches the system to "Adjust Scope" mode, and centers the telescope image of Betelgeuse in the graphical cross-hair image. The system is then taken out of Adjust Scope mode, switched back to a wide-field view, and adjusted to center the Pleiades cluster (Plate 9.3-4). The observer then switches to a 5-degree field (Plate 9.3-5).

To identify the star SAO 76136, the observer selects the search function and types in the designation (Plate 9.3-6). The TGDS centers the correct star in the image (Plate 9.3-7). The observer then sets the system to telescope view mode (Plate 9.3-8), and the

telescope automatically slews towards the star. When the observer looks into the eyepiece, the image of the actual star appears very close to the graphics image of the star, providing a positive identification. This may be important since the asteroid occulting the star is often too faint to be seen, and positive identification of the correct star is essential to performing the observation.

When moving on to another object for some other type of observation, the observer need only make a small adjustment to the initial position of the telescope (if required). The telescope will move to any star which the operator centers in the screen. As can be seen, this system makes the operation of the telescope extremely simple.

Plate 9.3-1 Initial Display

Plate 9.3-2 Centering Betelgeuse

Plate 9.3-3 Switching to Telescope View Mode

Plate 9.3-4 Centering the Pleiades Cluster

Plate 9.3-5 Larger View of Pleiades

Plate 9.3-6 Entering Designation of Target Star

Plate 9.3-7 Star Centered

Plate 9.3-8 Telescope View of SAO 76136

2 of/de 2

WE MAKE IT BETTER

METAL

TECHNOLOGY

10. CONCLUSIONS

The Telescope Graphics Display System was proposed as a means to increase the flexibility and ease-of-use of smaller observatory instruments. As such, it could potentially increase the productivity of valuable resources.

A prototype system was designed and constructed to prove the feasibility of the concepts. It was designed to meet the limited goal of locating and identifying stars, both for use as an observing aid and a training device. The performance of the prototype system was quite adequate to demonstrate the viability of the technique.

The relatively low-cost of the system, as an add-on to existing "off-the-shelf" telescopes, places it within the range of the smaller university observatories and even serious amateur astronomers. Many amateurs provide valuable observations to the professional community. The TGDS could potentially improve the quantity and quality of these observations. A large number of potential applications were identified, ranging from training to the performance of difficult measurements.

There are many possibilities for further work on the system. These could be divided into hardware enhancements and software enhancements.

The optical assembly could potentially be made smaller and more lightweight. The amount of extra focal length required could be reduced at the same time. A lever could also be installed to permit switching between the combined telescope/graphics display, and a telescope-only mode. This would eliminate the degradation in the image caused by the partially-aluminized mirror. As mentioned earlier, a specially-manufactured CRT could provide higher-resolution displays, perhaps even with colour and an interface to the new VGA standard display cards.

The single most important hardware improvement can be made to the telescope mounting. A more stable telescope mounting with less backlash in the motor trains would be highly desirable. An absolute-position measuring optical shaft encoder on each axis

would provide direct position feedback for closed-loop control. Off-the-shelf optical encoder technology for telescopes has recently become available with the introduction of an intelligent telescope setting-circle system by Advanced Technology Instruments.

The user interface could be improved by adding a small keypad which could be easily hand-held. This would make the operation of the system much easier at the telescope eyepiece.

The portability of the system could be improved by constructing a custom single-board computer with hard drive and appropriate graphics display. Using existing single-board computers, and a 5-inch CRT, an extremely compact computer could be built.

As for software improvements, there are a number of possibilities. The speed of loading and displaying the star maps could likely be improved. Additional databases could be incorporated, including non-stellar objects. The software could be designed to accommodate errors in the polar alignment and flexure in the mounting by analyzing position errors whenever the system is realigned. Many additional functions could be added in order to facilitate more of the proposed applications.

As can be seen, the hardware exists to make the TGDS a commercially viable system. The existence of a market is shown by the number of computer-aided telescope and software products being advertised in astronomy magazines. The first novel concept of the TGDS – the real-time display of annotated star charts in the vicinity of the telescope field – is almost certainly a practical product in the very near term. The second novel concept – the eyepiece graphics display – will likely be viable as a high-end product with some further refinement.

A. CALCULATIONS

A.1 CYLINDRICAL PROJECTION

This map projection is used to calculate the X-Y coordinates of stars near the celestial equator.

The Right Ascension and Declination are approximately linear near the equator. Thus, the following values are assigned:

$$\begin{aligned} SF &= 100 * \pi / 2 / \text{Scale} \\ X &= (\text{Dec}_s - \text{Dec}_m) * SF \\ Y &= (\text{RA}_s - \text{RA}_m) * SF \end{aligned}$$

Where:

Scale = Field of View in radians
SF = Scale Factor
Dec_s = Declination of Star
Dec_m = Declination of Map Center
RA_s = Right Ascension of Star
RA_m = Right Ascension of Map Center
X = X Coordinate
Y = Y Coordinate

Thus, the image is scaled so that at a scale of 180 degrees, an entire hemisphere of the sky appears 100 units across. This number is further scaled to the screen size before displaying.

A.2 LAMBERT CONFORMAL CONIC PROJECTION

This map projection is used to calculate the X-Y coordinates of stars not near the celestial equator. Initially, the Scale_Factor is calculated as for the Cylindrical projection:

$$SF = 100 * \pi / 2 / \text{Scale}$$

The center of the map and two reference lines (where the cone intersects the sphere) are taken as follows:

$$\begin{aligned}\text{Phi}_0 &= \text{Dec}_m \\ \text{Phi}_1 &= \text{Dec}_m + \text{Scale}/4 \\ \text{Phi}_2 &= \text{Dec}_m - \text{Scale}/4\end{aligned}$$

Where:

$$\begin{aligned}\text{Phi}_0 &= \text{Center of Map} \\ \text{Phi}_1 &= \text{Top Intersection Line} \\ \text{Phi}_2 &= \text{Bottom Intersection Line}\end{aligned}$$

The following values are then calculated:

$$\begin{aligned}N &= \frac{\ln(\cos(\text{Phi}_1) / \cos(\text{Phi}_2))}{\ln(\tan(\pi/4 + \text{Phi}_2/2) / \tan(\pi/4 + \text{Phi}_1/2))} \\ F &= (\cos(\text{Phi}_1) * \tan(\pi/4 + \text{Phi}_1/2))^N * SF \\ \text{Rho}_0 &= \frac{F}{(\tan(\pi/4 + \text{Phi}_0/2))^N}\end{aligned}$$

Each star is then processed as follows:

$$\begin{aligned}\text{Theta} &= (\text{RA}_s - \text{RA}_m) * N \\ \text{Rho} &= \frac{F}{(\tan(\pi/4 + \text{Dec}/2))^N} \\ X &= \text{Rho} * \sin(\text{Theta}) \\ Y &= \text{Rho}_0 - \text{Rho} * \cos(\text{Theta})\end{aligned}$$

Where:

Dec_s = Declination of Star
 Dec_m = Declination of Map Center
 RA_s = Right Ascension of Star
 RA_m = Right Ascension of Map Center
Scale = Field of View in radians

Since the projection is always adjusted to be most accurate near the center of the map, and the scale is small in the telescope eyepiece, there is no observable error.

A.3 ALTAZIMUTH ROTATION

This map projection is used to rotate the sky map to an altitude-azimuth coordinate system. This is used to show the stars near the horizon as seen by an observer on the ground.

$$\text{Alt} = \arcsin \left(\sin(\text{Lat}) * \sin(\text{Dec}) + \cos(\text{Lat}) * \cos(\text{Dec}) * \cos(\text{HA}) \right)$$

$$\text{Az} = \arccos \left[\frac{(\sin(\text{Dec}) - \sin(\text{Lat}) * \sin(\text{Alt}))}{(\cos(\text{Lat}) * \cos(\text{Alt}))} \right]$$

Where:

Lat = Latitude of Observer
 HA = Hour Angle of Star (= Sidereal Time - RA)
 Dec = Declination of Star
 Alt = Altitude of Star
 Az = Azimuth of Star

A.4 MOTOR STEP SIZE

The distance the telescope moves with each step is calculated from the following data (obtained empirically due to the lack of documentation on the motor drive):

4 full-steps or 8 half-steps per complete motor cycle
 12 cycles per revolution, ie. 7.5 degrees per full step
 30:1 gear reduction in motor gearbox
 144:1 reduction in worm gear

Therefore:

$12 * 30 * 144 = 51,840$ cycles per telescope revolution
 $51,840 * 4 = 207,360$ steps per telescope revolution
 $207,360 * 2 = 414,720$ half-steps per telescope revolution

A.5 SIDEREAL TIME

The time between interrupts for the sidereal time clock is calculated from the following data:

CPU Clock Rate = 4.096 MHz

E-Clock Rate = 1.024 MHz

One Telescope Revolution = 414,720 steps

One Sidereal Day = 86164.1 seconds

We need one telescope revolution per sidereal day. Therefore:

Sidereal rate = 0.2077645 seconds per half-step
 = 212,751 cycles at 1.024 MHz

With a software divide-by-four, we need the 6840 timer chip to divide the 1.024 MHz E clock by:

$$212,751 / 4 = 53,188$$

The rounding error results in a drift of approximately 0.47 seconds per day. In the telescope, the drive error will be about 7.05 seconds of arc, or about 2 half-steps per day.

A.6 COORDINATE PRECESSION

In order to precess all of the databases to the same epoch, the following formulae were used [5]:

$$RA_{Temp} = [3.07327 + 1.33617 * \sin(RA) * \tan(Dec)] \\ * (Epoch_2 - Epoch_1) * \pi / 43200$$

$$Dec = Dec + 20.0426 * \cos(RA) \\ * (Epoch_2 - Epoch_1) * \pi / 648000$$

$$RA = RA_{Temp} + RA$$

Where:

RA = Right Ascension
 Dec = Declination
 Epoch₂ = Julian Date of Destination Epoch in days
 Epoch₁ = Julian Date of Starting Epoch in days

These equations involve some approximations to obtain reasonable processing speed and are accurate to within a few seconds of arc. Since the telescope is initially aligned to a star, the relative error over reasonable slewing distances is much smaller. The errors are therefore insignificant compared to mechanical inaccuracies.

B. REFERENCES

1. Genet, Russel M., et. al., **The Automatic Photoelectric Telescope Service**, Center for Astrophysics Preparing Series No. 2476, Cambridge Massachusetts, 1987.
2. Taff, Lawrence G., **Celestial Mechanics**, John Wiley & Sons, Inc., Toronto, Ontario, 1985.
3. Sinot, Roger W., "Astronomical Computing", **Sky and Telescope**, July 1986, Sky Publishing Corp., Cambridge, Mass., 1986.
4. Manly, Peter L., "Applications of Microcomputers in Astronomy", **Microcomputers in Astronomy II**, Russel M. Genet, Karen A. Genet, editors, Fairborn Observatory, Fairborn, Ohio, 1984.
5. Trueblood, Mark, and Russel Genet, **Microcomputer Control of Telescopes**, Willmann-Bell Inc., Richmond, Virginia, 1985.
6. Ghedini., Silvano, **Software for Photoelectric Astronomy**, Willmann-Bell Inc., Richmond, Virginia, 1982.
7. Genet, Russel M., editor, **Solar System Photometry Handbook**, Willmann-Bell, Inc., Richmond, Virginia, 1983.
8. Edberg, Stephen J., **International Halley Watch Amateur Observers' Manual for Scientific Comet Studies**, Jet Propulsion Laboratory of the California Institute of Technology, 1983.
9. Harvey E. White, **Modern College Physics**, sixth edition, D. Van Nostrand Company, Toronto, Ontario, 1972.

10. **Stepper Motor Handbook**, North American Philips Controls Corp., USA, 1979.

C. BIBLIOGRAPHY

Anderson, Ed, **FINDER and ASTRO on the VMS VAX**, National Optical Astronomy Observatories, Tucson, Arizona, 1986.

Borland International, **Turbo Pascal Reference Manual**, Borland International, Scotts Valley, California, 1984.

Borland International, **Turbo Graphix Toolbox Owner's Handbook**, version 1.0, Borland International, Scotts Valley, California, 1985.

Duffett-Smith, Peter, **Astronomy with your Personal Computer**, Cambridge University Press, Cambridge, 1985.

Duffett-Smith, Peter, **Practical Astronomy with your Calculator**, second edition, Cambridge University Press, Cambridge, 1981.

Genet, Russel M., editor, **Microcomputers in Astronomy**, Fairborn Observatory, Fairborn, Ohio, 1983.

Genet, Russel M. , and Karen A. Genet, editors, **Microcomputers in Astronomy II**, Fairborn Observatory, Fairborn, Ohio, 1984.

Leventhal, Lance A., **6809 Assembly Language Programming**, Osborne/McGraw-Hill, Berkeley, California, 1981.

Motorola, Inc., **The Complete Motorola Microcomputer Data Library**, Motorola Inc., USA, 1978.

Scanlon, Leo J., **IBM PC Assembly Language Programming, A Guide for Programmers**, Robert J. Brady Company, Bowie, Maryland, 1983.

Sherrod, P. Clay, **A Complete Manual of Amateur Astronomy**, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1981.

**Trueblood, Mark, and Russel Genet, Microcomputer Control of Telescopes.
Willmann-Bell Inc., Richmond, Virginia, 1985.**

D. SOFTWARE LISTINGS

Main Graphics Program

Name: Scope

Location: SCOPE.PAS

Purpose: Generates star maps for telescope display system

Copyright (c) 1986, 1987 by
Doug George, 66 Shetland Way,
Kanata, Ontario, K2M 1S4

```
Program Map ( input, output );
{SC-}
{SI global.typ}
{  disptec.pas} { Select for TECMAR }
{  dispcqa.pas} { Select for IBM COLOUR }
{  dispeqa.pas} { Select for EGA WONDER }
{SI convert.pas}
{SI input.pas}
{SI readfile.pas}
{SI initial.pas}
{SI draw.pas}
{SI menu.pas}

{ Mainline }
begin

{ initialize }
Initialize;

Main_Menu;

TextMode ( 2 );
end.
```

```

-----
Name:      Global_Types
Location:  GLOBAL.TYP
Purpose:   Global variables for map-drawing program

Copyright (c) 1986, 1987 by
Doug George, 66 Shetland Way
Kanata, Ontario, K2M 1S4
-----

```

```

( *** Global Constants *** )

```

```

const
  Max_Stars      = 2621;
  Max_Name       = 6;
  Max_String     = 80;
  Max_Time_String = 8;
  Max_Dec_String = 9;
  Max_Date_String = 10;
  Max_Control_String = 4;
  Max_Option     = 8;
  Max_Pages      = 5;
  Max_SAO_Files  = 102;
  Max_YBSC_Files = 4;
  Backlash_Distance = 38;
  Backlash_Delay  = 1400;
  Max_Slow_Slew   = 500;
  RA_Steps_Per_Radian = 6.600473802E4;
  Dec_Steps_Per_Radian = 6.600473802E4;
  Default_Telescope_Scale = 0.32; { In degrees }

```

```

( *** Global Types *** )

```

```

type
  { Registers for MS-DOS or BIOS calls :
  Registers =
    record
      AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags : integer;
    end;

```

```

  { String and I/O Types :

```

```

  String_Type = String[ Max_String ];
  Control_String_Type = String[ Max_Control_String ];
  Name_Type = String[ Max_Name ];

```

```

  { Time Types }
  Time_Type = String [ Max_Time_String ];
  Dec_Type = String [ Max_Dec_String ];
  Date_Type = String [ Max_Date_String ];

```

```

  Prompt_Mode_Type =
    ( Integer_Prompt,
      Real_Prompt,
      String_Prompt,
      Boolean_Prompt );

```

```

  Prompt_Data_Type =
    record
      case Prompt_Mode : Prompt_Mode_Type of
        Integer_Prompt :
          ( Minimum_Integer : integer;
            Maximum_Integer : integer;
          );
        Real_Prompt :
          ( Minimum_Real : real;
            Maximum_Real : real;
          );
      end;
    end;

```

```

    );
    String_Prompt :
    (
        Max_Length : integer;
    );
    Boolean_Prompt :
    ();
end;

Option_Type = packed array [ 0 .. Max_Option ] of String_Type;

Menu_Type =
record
    Max      : 1 .. Max_Option;
    Title    : String_Type;
    Option   : Option_Type;
end;

( These types describe the objects seen )

Object_Type = ( Star, Nebula, Galaxy, Planetary,
                Open_Cluster, Globular_Cluster,
                Moon, Planet, Asteroid, Other, None );

( Character Matrix )
FontChar = array [ 0 .. 13 ] of integer;
Display_Font = array [ 0 .. 255 ] of FontChar;

( Star Data Array )
Star_Data_Type =
record
    Name : Name_Type;
    RA   : real;
    Dec  : real;
    Mag  : integer;
end;

Star_Array_Type = packed array [ 1 .. Max_Stars ] of Star_Data_Type;

Star_Table_Record =
record
    First_Star      : 0 .. Max_Stars;    ( Present first star )
    Last_Star       : 0 .. Max_Stars;    ( Present last star )
    Number_Of_Stars : 0 .. Max_Stars;    ( Permanent last star )
    Star            : Star_Array_Type;
end;

Star_Table_Type = ^Star_Table_Record;

Star_Table_Array_Type = array [ 1 .. Max_Pages ] of Star_Table_Type;

SAO_RA_Table_Type = array [ 0 .. Max_SAO_Files ] of real;
YBSC_RA_Table_Type = array [ 0 .. Max_YBSC_Files ] of real;

( Projection types )

Projection_Type = ( Athmuthal_Stereographic, Lambert_Conformal,
                   Conic, Cylindrical, No_Projection );

Map_Data_Type =
record
    R      : real;
    Phi_0  : real;
    Phi_1  : real;
    Phi_2  : real;
    Lambda_0 : real;
    N      : real;
    F      : real;
    Rho_0   : real;
    Cos_Phi_0 : real;
    Sin_Phi_1 : real;
    Cos_Phi_1 : real;

```

```

    Sin_Latitude : real;
    Cos_Latitude : real;
end;

Initial_Data_Type =
record
    Projection : Projection_Type;
    Alt_Az_Mode : boolean;
    Scale : real;
    Right_Ascension : real;
    Declination : real;
    X_Offset : real;
    Y_Offset : real;
end;

{ Global data area definitions.
{ The global data area stores variables used in the display
{ routines which must be saved between successive invocations.

{ Global Variables }
var
    Star_Table      : Star_Table_Array_Type;
    Font            : Display_Font;

    Deg_To_Rad      : real;
    Rad_To_Deg      : real;
    Rad_To_Hour      : real;
    Hour_To_Rad      : real;
    Two_Pi          : real;
    Half_Pi          : real;
    Quarter_Pi       : real;

    Date_String      : Date_Type;
    Time_String      : Time_Type;
    Siderial_String  : Time_Type;
    Time             : real;
    Siderial         : real;
    Time_Zone        : real;
    Telescope_Scale  : real;
    X_Scale          : real;
    Y_Scale          : real;

    Latitude         : real;
    Longitude        : real;

    Map_Data         : Map_Data_Type;
    Initial_Data     : Initial_Data_Type;

    Magnitude_Offset : integer;
    Display_Bank     : integer;
    Update_Bank      : integer;

    SAO_RA_Table     : SAO_RA_Table_Type;
    YBSC_RA_Table    : YBSC_RA_Table_Type;

```

```

-----
Name:      Display_Driver
Location:   DISPEGA.PAS
Purpose:    Extended Colour Graphics Adaptor display
            drivers for star map program
Copyright (c) 1986, 1987 by
Doug George, 66 Shetland Way
Kanata, Ontario, K2M 1S4
-----
( Global constants for screen size )

const
    Left_Margin      = 0;
    Right_Margin     = 639;
    Top_Margin        = 0;
    Bottom_Margin     = 349;
    X_Center          = 320;
    Y_Center          = 175;
    Initial_X_Scale   = 2.14;
    Initial_Y_Scale   = 1.7;

Type
    Spot_Array_Type = array[ 1 .. 3 ] of integer;

( Local Permanent Variables )

var
    Column_Byte_Lookup : array[ Left_Margin .. Right_Margin ] of integer;
    Column_Bit_Lookup  : array[ Left_Margin .. Right_Margin ] of integer;
    Row_Byte_Lookup    : array[ Top_Margin .. Bottom_Margin ] of integer;

( *** Screen Manipulation Routines *** )

procedure Clear_Screen;

    procedure CLR0;
        external 'CLR0.BIN';
    procedure CLR1;
        external 'CLR1.BIN';

begin
    if Update_Bank = 0 then
        CLR0
    else
        CLR1;

    (*
begin
    Port[ $3CE ] := 0;
    Port[ $3CF ] := 0;
    Port[ $3CE ] := 1;
    Port[ $3CF ] := 0;
    Port[ $3CE ] := 8;
    Port[ $3CF ] := $FF;
    if Update_Bank = 0 then
        FillChar ( Mem[ $A000:0000 ], 32768, 0 )
    else
        FillChar ( Mem[ $A000:0000 ], 32768, 0 );
    *)
end;

procedure Set_Video_Mode(
    bi : integer;
    bh : integer;

```

```

        mode : integer );

type
  Registers =
    record
      AX, BX, CX, DX, BP, SI, DI, DS, ES, FLAGS : integer;
    end;

var
  Regs:Registers;

begin
  with Regs do
    begin
      AX := mode;
      BX := 256 * bh + bl;
      Intr( $10, REGS );
    end;
  end; { Set_Video_Mode }

procedure Graphics_Out(
  Index : integer;
  Value : integer );
begin
  { local procedure }
  procedure OUTX(
    var Port : integer;
    var Value : integer );
  external 'OUTX.BIN';

  var
    Port : integer;
    Value_Out : integer;

  begin
    Port := $3CE;
    Value_Out := Index;
    OUTX( Port, Value_Out );

    Port := $3CF;
    Value_Out := Value;
    OUTX( Port, Value_Out );

  end;

procedure Set_Graphics_Mode;
const
  Colour_Map : array [ 0 .. 15 ] of integer =
    ( 0, 8, 2, 32, 3, 33, 42, 15, 39, 47, 14, 30, 46, 34, 17, 43 );

var
  I : integer;

begin
  GraphMode;

  { Set video to mode 16 }
  Set_Video_Mode( 0, 0, 16 );

  { Write in the new palette }
  for I := 0 to 15 do
    Set_Video_Mode( I, Colour_Map[ I ], $1000 );

  { Row look-up table }
  for I := Top_Margin to Bottom_Margin do
    Row_Byte_Lookup[ I ] := (Right_Margin + 1) div 8 * I;

  { Column look-up tables }

```

```

for I := Left_Margin to Right_Margin do
begin
  Column[Byte_Lookup[ I ]] := I shr 3;
  Column[Bit_Lookup[ I ]] := 128 shr (I mod 8);
end;

(* Set and/or mode *)
Graphics_Out( 3, 0 );

(* Enable set/reset *)
Graphics_Out( 1, $0F );

Display_Bank := 0;
Update_Bank := 0;
end;

procedure Set_Text_Mode;
begin
  (* Set/reset to normal *)
  Graphics_Out( 0, 0 );

  (* Set/reset enable to normal *)
  Graphics_Out( 1, 0 );

  (* Bit mask to normal *)
  Graphics_Out( 8, $FF );

  (* Reset video mode *)
  Set_Video_Mode( 0, 0, 3 );

  Text_Mode( 0 );
  Text_Color( 2 );
end;

(* Toggle memory banks *)
procedure Toggle_Banks;
begin
  if Update_Bank = 0 then
  begin
    Display_Bank := 0;
    Update_Bank := 1;
    Set_Video_Mode( 0, 0, 1280 );
  end
  else
  begin
    Display_Bank := 1;
    Update_Bank := 0;
    Set_Video_Mode( 0, 0, 1281 );
  end;
end;

(* Procedures to output to screen *)
procedure Spot0(
  var Spot_Array : Spot_Array_Type );
external 'SPOT0.BIN';

procedure Spot1(
  var Spot_Array : Spot_Array_Type );
external 'SPOT1.BIN';

(* Draw crosshairs *)
procedure Crosshairs (
  X_Location : integer;
  Y_Location : integer;
  Brightness : integer );

(* Local procedure to output a spot *)
procedure Spot (

```

```

X      : integer;
Y      : integer );

var
  Spot_Array : Spot_Array_Type;

begin
  if (X < Right_Margin) and (X > Left_Margin) and
     (Y > Top_Margin) and (Y < Bottom_Margin) then
    begin
      Spot_Array[ 1 ] := Brightness;
      Spot_Array[ 2 ] := Column_Bit_Lookup[ X ];
      Spot_Array[ 3 ] := Row_Byte_Lookup[ Y ] + Column_Byte_Lookup[ X ];
      if Update_Bank = 0 then
        Spot0[ Spot_Array ]
      else
        Spot1[ Spot_Array ];
      end;
    end;
end;

var
  I : integer;

begin
  for I := X_Location - 20 to X_Location + 10 do
    Spot ( I, Y_Location );
  for I := X_Location + 10 to X_Location + 20 do
    Spot ( I, Y_Location );
  for I := Y_Location - 20 to Y_Location + 10 do
    Spot ( X_Location, I );
  for I := Y_Location + 10 to Y_Location + 20 do
    Spot ( X_Location, I );
  end;

  ( Display a point )
  procedure Display(
    X_Real : real;
    Y_Real : real;
    Magnitude : integer;
    Cross : boolean );

  var
    Spot_Array : Spot_Array_Type;
    X : integer;
    Y : integer;

  begin
    X_Real := X_Real - Initial_Data.X_Offset;
    Y_Real := Y_Real - Initial_Data.Y_Offset;

    if (X_Real > Left_Margin) and (X_Real < Right_Margin) and
       (Y_Real > Top_Margin) and (Y_Real < Bottom_Margin) then
      begin
        X := Round ( X_Real );
        Y := Round ( Y_Real );

        Magnitude := (650 - Magnitude - Magnitude_Offset) div 10;

        if Cross then
          Magnitude := Magnitude - 2;

        if Magnitude > 15 then
          Magnitude := 15
        else if Magnitude < 1 then
          Magnitude := 1;

        if Cross then
          Crosshairs ( X, Y, Magnitude )
        else
          begin

```



```

Spot_Array[ 1 ] := Magnitude;
Spot_Array[ 2 ] := Column_Bit_Lookup[ X ];
Spot_Array[ 3 ] := Row_Byte_Lookup[ Y ] + Column_Byte_Lookup[ X ];
if Update_Bank = 0 then
    Spot0( Spot_Array )
else
    Spot1( Spot_Array );
end;
end;
end;

```

• This procedure displays a character on the screen •

```

procedure Write_Text (
    Text_String: String_Type ;
    X           : integer ;
    Y           : integer );

```

• This local procedure outputs one line of a character •

```

procedure Line_Out (
    Char_Slice : integer ;
    X           : integer ;
    Y           : integer );

```

```

var
    Spot_Array : Spot_Array_Type;

```

```

const
    Brightness = 7;

```

```

begin

```

```

    Spot_Array[ 1 ] := Brightness;
    Spot_Array[ 2 ] := Char_Slice;
    Spot_Array[ 3 ] := Row_Byte_Lookup[ Y ] + Column_Byte_Lookup[ X ];
    if Update_Bank = 0 then
        Spot0( Spot_Array )
    else
        Spot1( Spot_Array );
    end;
end;

```

• Local variables for Write_Text •

```

var
    I       : integer;
    J       : integer;
    Slice   : integer;
    X_Pos   : integer;
    Y_Pos   : integer;

```

```

begin

```

• Loop for each character in string •

```

for J := 1 to length ( Text_String ) do
    begin

```

• Loop for each line in character •

```

        for I := 0 to 13 do
            begin

```

```

                Slice := Font_Table ( Text_String [ J ] [ I ] );
                X_Pos := 16 * J + X;
                Y_Pos := I * 2 + Y;

```

```

                if (X_Pos > Left_Margin) and (X_Pos < Right_Margin) and
                   (Y_Pos > Top_Margin) and (Y_Pos < Bottom_Margin + 13) then
                    begin

```

```

                        ; Output right side ;
                        Line_Out ( Slice, X_Pos + 8, Y_Pos );

```

```

                        ; Output left side ;
                        Line_Out ( Slice shr 8, X_Pos, Y_Pos );
                    end;
            end;
        end;
    end;
end;

```

```

        { Left side, second line }
        Line_Out ( Slice shr 8, X_Pos, Y_Pos + 1 );

        { Right side, second line }
        Line_Out ( Slice, X_Pos + 8, Y_Pos + 1 );
        end;
    end;
end;

```

```

{ Read in Font }
procedure Initialize_Font;

var
    FontFile:file of Display_Font;

begin
    { Initialize the Font memory }
    assign ( FontFile, 'ASTRO.FON' );
    reset ( FontFile );
    read ( FontFile, Font );
    close ( FontFile );
end;

```

```

-----
Name:      Conversion routines
Location:  CONVERT.PAS
Purpose:   Coordinate conversion routines for map-drawing

Copyright (c) 1986, 1987 by
Doug George, 66 Shetland Way
Kanata, Ontario, K2M 1S4
-----

(* Raise variable to a power *)
function Power ( X, Y : real ) : real;
begin
  Power := exp ( Y * ln ( X ) );
end;

(* Calculate Tangent *)
function Tan ( X : real ) : real;
begin
  Tan := sin ( X ) / cos ( X );
end;

(* Calculate ArcSin *)
function ArcSin ( X : real ) : real;
begin
  if abs ( X ) < 1.0 then
    ArcSin := ArcTan ( X / Sqrt ( 1.0 - Sqr ( X ) ) )
  else if X < 0.0 then
    ArcSin := -Half_Pi;
  else
    ArcSin := Half_Pi;
  end;
end;

(* Calculate ArcCos *)
function ArcCos ( X : real ) : real;
begin
  if abs ( X ) < 1.0 then
    ArcCos := -ArcTan ( X / Sqrt ( 1.0 - Sqr ( X ) ) ) + Half_Pi;
  else if X < 0.0 then
    ArcCos := Pi;
  else
    ArcCos := 0.0;
  end;
end;

(* Initialize the Athmuthal Stereographic variables *)
procedure Pre_Athmuthal_Stereographic;

begin
  with Map_Data, Initial_Data do
    begin
      Lambda_0 := Right_Ascension;
      R := 100.0 * Pi / Scale;
      Phi_0 := 0.0;
      Phi_1 := 0.0;
      Sin_Phi_1 := sin ( Phi_1 );
      Cos_Phi_1 := cos ( Phi_1 );
    end;
  end;

(* Calculate the Athmuthal Stereographic projection *)
procedure Calc_Athmuthal_Stereographic (
  Lambda, Phi : real;
  var X, Y : real );

var
  K, L : real;
  Delta_Lambda : real;

```

```

Sin_Delta_Lambda : real;
Cos_Delta_Lambda : real;
Sin_Phi : real;
Cos_Phi : real;
Sin_Phi_1 : real;
Cos_Phi_1 : real;
Cos_Phi_Cos_Delta_Lambda : real;
R_K_0 : real;

begin
with Map_Data do
begin
( Calculate Delta Lambda )
Delta_Lambda := Lambda - Lambda_0;
if Delta_Lambda > Pi then
Delta_Lambda := Delta_Lambda - Two_Pi;
else if Delta_Lambda < - Pi then
Delta_Lambda := Delta_Lambda + Two_Pi;

( Output star if it is inside hemisphere )
if abs ( Delta_Lambda ) <= Half_Pi then
begin

Sin_Delta_Lambda := sin ( Delta_Lambda );
Cos_Delta_Lambda := cos ( Delta_Lambda );
Sin_Phi := sin ( Phi );
Cos_Phi := cos ( Phi );
Cos_Phi_Cos_Delta_Lambda := Cos_Phi * Cos_Delta_Lambda;

R_K_0 := R / ( 1.0 + Sin_Phi_1 * Sin_Phi
               + Cos_Phi_1 * Cos_Delta_Lambda );
X := R_K_0 * Cos_Phi * Sin_Delta_Lambda;
Y := R_K_0 * Cos_Phi_1 * Sin_Phi -
     Sin_Phi_1 * Cos_Phi_Cos_Delta_Lambda;
end
else
begin
X := MaxInt;
Y := MaxInt;
end;
end;
end;

( Initialize the Lambert Conformal Conic variables )
procedure Pre_Lambert_Conformal;

begin
with Map_Data, Initial_Data do
begin
Lambda_0 := Right_Ascension;
R := 100.0 * Half_Pi / Scale;
Phi_0 := Declination;
Phi_1 := Declination + Scale / 4.0;
Phi_2 := Declination - Scale / 4.0;

N := ln ( cos ( Phi_1 ) / cos ( Phi_2 ) )
     / ln ( tan ( Quarter_Pi + Phi_2 / 2.0 ) /
           tan ( Quarter_Pi + Phi_1 / 2.0 ) );
F := Power ( cos ( Phi_1 ) * tan ( Quarter_Pi + Phi_1 / 2.0 ), N );

( Let A = R * F )
R := R * F;
Rho_0 := R / Power ( tan ( Quarter_Pi + Phi_0 / 2.0 ), N );

end;
end;

( Calculate the Athmuthal Stereographic projection )
procedure Calc_Lambert_Conformal (

```

```

    Lambda, Phi : real;
    var X, Y : real );

*end*
    Modified := false;

var
    Delta_Lambda : real;
    Rho : real;
    Theta : real;
    In_Range : boolean;

begin
    with Map_Data do
        begin
            if Phi > 0. then
                In_Range := (Phi > 0.0) and (Phi < Half_Pi)
            else
                In_Range := (Phi < 0.0) and (Phi > -Half_Pi);

            if In_Range then
                begin
                    Delta_Lambda := Lambda - Lambda_0;
                    if Delta_Lambda > Pi then
                        Delta_Lambda := Delta_Lambda - Two_Pi;
                    else if Delta_Lambda < -Pi then
                        Delta_Lambda := Delta_Lambda + Two_Pi;

                    Rho := R / Power ( tan ( Quarter_Pi + Phi / 2.0 ), N );
                    Theta := N * Delta_Lambda;
                    X := Rho * sin ( Theta );
                    Y := Rho * cos ( Theta );
                end
            else
                begin
                    X := MaxInt;
                    Y := MaxInt;
                end;
            end;

        end;
    end;

    (*      NEW VERSION - NOT IMPLEMENTED
    NOTE: RANGE CHECKS MUST BE INCLUDED
    procedure Lambert_Conformal (
        Lambda, Phi : real;
        var X, Y : real );

    var
        Data : array [ 1 .. 7 ] of real;
        In_Range : boolean;

    begin
        with Map_Data do
            begin
                Data[ 1 ] := R;
                Data[ 2 ] := Phi;
                Data[ 3 ] := N;
                Data[ 4 ] := Delta_Lambda;
                Data[ 5 ] := Rho_0;
                FASTLAM ( Data );
                X := Data[ 6 ];
                Y := Data[ 7 ];
            end;
        end;

        (* Initialize the Conic variables *)
        procedure Pre_Conic;
        begin
            with Map_Data, Initial_Data do
                begin

```

```

    end;
end;

( Calculate the Conic projection )
procedure Calc_Conic (
    Lambda, Phi : real;
    var X, Y : real );

begin
    X := MaxInt;
    Y := MaxInt;
end;

( Initialize the Cylindrical variables )
procedure Pre_Cylindrical;

begin
    with Map_Data, Initial_Data do
        begin
            Lambda_0 := Right_Ascension;
            Phi_0 := Declination;
            R := 100.0 * Half_PI / Scale;
        end;
    end;

( Calculate the Cylindrical projection )
procedure Calc_Cylindrical (
    Lambda, Phi : real;
    var X, Y : real );

begin
    with Map_Data do
        begin
            X := Lambda - Lambda_0;
            Y := Phi - Phi_0;

            if X > Pi then
                X := X - Two_Pi;
            else if X < -Pi then
                X := X + Two_Pi;
            if Y > Half_Pi then
                Y := Y - Pi;
            else if Y < -Half_Pi then
                Y := Y + Pi;

            X := X * R;
            Y := Y * R;

        end;
    end;

( Convert to Altazimuth Coordinates )
procedure Altazimuth (
    HA, Dec : real;
    var Az, Alt : real );

var
    Sin_Dec : real;
    Sin_HA : real;
    Sin_Alt : real;
    Cos_Alt : real;

begin
    with Map_Data do
        begin

```

```

Sin_Dec := sin ( Dec );
Sin_HA := sin ( HA );

Sin_Alt := Sin_Latitude * Sin_Dec +
           Cos_Latitude * cos ( Dec ) * cos ( HA );

Alt := ArcSin ( Sin_Alt );
Cos_Alt := cos ( Alt );

if Cos_Alt <> 0.0 then
    Az := ArcCos ( ( Sin_Dec - Sin_Latitude * Sin_Alt ) /
                  ( Cos_Latitude * Cos_Alt ) );
else
    Az := MaxInt;
endif

if Sin_HA > 0.0 then
    Az := Two_Pi - Az;
endif;
end;

(*      NEW VERSION - NOT IMPLEMENTED
      NOTE: RANGE CHECKS MUST BE INCLUDED
*)

(* Convert to Altazimuth Coordinates *)
procedure Altazimuth (
    HA, Dec : real;
    var Az, Alt : real );

var
    Data : array [ 1 .. 6 ] of real;

begin
    with Map_Data do
        begin
            Data[ 1 ] := Sin_Latitude;
            Data[ 2 ] := Cos_Latitude;
            Data[ 3 ] := HA;
            Data[ 4 ] := Dec;
            FASTALT ( Data );
            Az := Data[ 5 ];
            Alt := Data[ 6 ];
        end;
    end;
end;

*)

(* Convert to Equatorial Coordinates *)
procedure Altazimuth (
    Az, Alt : real;
    var HA, Dec : real );

begin
    As above procedure, except reverse calling sequence as indicated.
end;

*)

(* Perform initial calculations *)
procedure Pre_Calculate;

begin
    (* Reset offsets *)
    Initial_Data.X_Offset := 0.0;
    Initial_Data.Y_Offset := 0.0;

    (* Initialize projection *)

```

```

case Initial_Data.Projection of
    Athmuthal_Stereographic :
        Pre_Athmuthal_Stereographic;

    Lambert_Conformal :
        Pre_Lambert_Conformal;

    Conic :
        Pre_Conic;

    Cylindrical :
        Pre_Cylindrical;
end;

end;

{ Draw Point }
procedure Project_Point (
    var Lambda : real;
    var Phi : real );

var
    X : real;
    Y : real;

begin
    if Initial_Data.Alt_Az_Mode then
        Altazimuth ( Lambda - Sideral, Phi, Lambda, Phi );

    case Initial_Data.Projection of
        Athmuthal_Stereographic :
            Calc_Athmuthal_Stereographic ( Lambda, Phi, X, Y );

        Lambert_Conformal :
            Calc_Lambert_Conformal ( Lambda, Phi, X, Y );

        Conic :
            Calc_Conic ( Lambda, Phi, X, Y );

        Cylindrical :
            Calc_Cylindrical ( Lambda, Phi, X, Y );
        No_Projection :
            begin
                X := Lambda;
                Y := Phi;
            end;
    end;

    if Initial_Data.Projection <> No_Projection then
        begin
            Lambda := - X_Scale * X + X_Center;
            Phi := - Y_Scale * Y + Y_Center;
        end;

    if abs ( Lambda ) > MaxInt then
        Lambda := MaxInt;

    if abs ( Phi ) > MaxInt then
        Phi := MaxInt;

end;

{ Parse Date }
procedure Parse_Date (
    Date_In : String_Type;
    var Year : integer;
    var Month : integer;
    var Day : integer );

var
    Result : integer;

```



```

begin
  { It is assumed that the date has the correct format. }
  Val ( Copy ( Date_In, 1, 4 ), Year, Result );
  Val ( Copy ( Date_In, 6, 2 ), Month, Result );
  Val ( Copy ( Date_In, 9, 2 ), Day, Result );
end;

{ Generate Time String }
procedure Time_To_String (
  Time_In : real;
  var Time_S : Time_Type );

type
  Temp_Type = packed array [ 1 .. 2 ] of char;

var
  Temp : Temp_Type;

  { Local procedure to format digits }
  procedure Copy_Time (
    Time_In : real;
    var Digits : Temp_Type );
  begin
    Digits [ 1 ] := chr ( ord ( '0' ) + Trunc ( Time_In ) div 10 );
    Digits [ 2 ] := chr ( ord ( '0' ) + Trunc ( Time_In ) mod 10 );
  end;

begin
  Time_In := Time_In * Rad_To_Hour;

  { Add 1/2 second to force rounding }
  Time_In := Time_In + 1.3888888888E-4;

  { Hours }
  Copy_Time ( Time_In, Temp );
  Time_S := Temp;

  { Minutes }
  Time_In := Time_In - Int ( Time_In );
  Time_In := Time_In * 60.0;
  Copy_Time ( Time_In, Temp );
  Time_S := Time_S + ':' + Temp;

  { Seconds }
  Time_In := Time_In - Int ( Time_In );
  Time_In := Time_In * 60.0;
  Copy_Time ( Time_In, Temp );
  Time_S := Time_S + ':' + Temp;
end;

{ Generate Declination String }
procedure Dec_To_String (
  Dec_In : real;
  var Dec_S : Dec_Type );

type
  Temp_Type = packed array [ 1 .. 2 ] of char;

var
  Temp : Temp_Type;

  { Local procedure to format digits }
  procedure Copy_Dec (
    Dec_In : real;
    var Digits : Temp_Type );

```

```

begin
  Digits [ 1 ] := chr ( ord ( '0' ) + Trunc ( Dec_In ) div 10 );
  Digits [ 2 ] := chr ( ord ( '0' ) + Trunc ( Dec_In ) mod 10 );
end;

begin
Dec_In := Dec_In * Rad_To_Deg;

if Dec_In < 0 then
begin
  Dec_In := -Dec_In;
  Dec_S := '-';
end
else
  Dec_S := '+';

{ Add 1/2 second to force rounding }
Dec_In := Dec_In + 1.388888888E-4;

{ Degrees }
Copy_Dec ( Dec_In, Temp );
Dec_S := Dec_S + Temp;

{ Minutes }
Dec_In := Dec_In - Int ( Dec_In );
Dec_In := Dec_In * 60.0;
Copy_Dec ( Dec_In, Temp );
Dec_S := Dec_S + ':' + Temp;

{ Seconds }
Dec_In := Dec_In - Int ( Dec_In );
Dec_In := Dec_In * 60.0;
Copy_Dec ( Dec_In, Temp );
Dec_S := Dec_S + ':' + Temp;

end;

{ Determine whether Leap Year ;
function Leap_Year ( Year : integer ) : boolean;
begin
  Leap_Year := ((Year mod 4) = 0) and not
               ((Year div 400 = 400) = Year);
end;

{ Calculate Day Number }
procedure Day_Number (
  Date_In : String_Type;
  var Number : integer );

var
  Year : integer;
  Month : integer;
  Day : integer;

begin
  Parse_Date ( Date_In, Year, Month, Day );

  if Month <= 2 then
    begin
      Number := Month - 1;

      if Leap_Year ( Year ) then
        Number := (Number * 62) div 2
      else
        Number := (Number * 63) div 2;
      end
    else
      begin
        Number := (Month + 1) * 306 div 10;
        if Leap_Year ( Year ) then

```

```

        Number := Number - 62
    else
        Number := Number - 63;
    end;

Number := Number + Day;

end;

{ Calculate Julian Day Number }
procedure Julian_Day_Number (
    Greg      : String_Type;
    Time_In   : real;
    var Number : real );

var
    Year : integer;
    Month : integer;
    Day   : integer;
    A, B, C, D : real;

begin
    Parse_Date ( Greg, Year, Month, Day );

    Number := Int ( Day ) + Time_In / Two_PI;

    if Month <= 2 then
        begin
            Year := Year - 1;
            Month := Month + 12;
        end;

    A := Int ( Year div 100 );

    if (Year > 1582) or (Month > 10) or (Day > 15) then
        B := 2.0 - A + Int ( A / 4.0 )
    else
        B := 0;

    C := Int ( 365.25 * Int ( Year ) );

    D := Int ( 30.6001 * (Month + 1) );

    Number := B + C + D + Number + 1720994.5;

end;

{ Calculate Siderial Time }
procedure Calculate_Siderial (
    Date_In   : String_Type;
    Time_In   : real;
    Zone       : real;
    Longitude  : real;
    var IST    : real );

const
    A = 0.0657098;
    C = 1.002738;

var
    B      : real;
    JD     : real;
    GMT     : real;
    GST     : real;
    Temp    : real;
    Year    : integer;
    Month   : integer;
    Day     : integer;
    Number  : integer;
    Temp_Date : Date_Type;

begin

```

```

Parse_Date ( Date_In, Year, Month, Day );

( Calculate constant B )

Temp_Date := Copy ( Date_In, 1, 4 );
Temp_Date := Temp_Date + '/01/00';
Julian_Day_Number ( Temp_Date, 0.0, JD );

Temp := (JD - 2415020.0) / 36525.0;
Temp := 6.6460656 + 2400.051262 * Temp + 0.00002581 * Sqr ( Temp );

B := 24.0 * (Year - 1899.0) - Temp;

( Calculate GMT )
GMT := Time_In * Rad_To_Hour + Zone;

( Calculate GST )
Day_Number ( Date_In, Number );

GST := Number * A - B + GMT * C + 24.0;

( Correct to centre of time zone )
LST := GST - Longitude * Rad_To_Hour;

while LST > 24.0 do
    LST := LST - 24.0;

while LST < 0.0 do
    LST := LST + 24.0;

LST := LST * Hour_To_Rad;

end;

( Convert a single-digit number to a character )
function Conv ( Digit : integer ) : char;
begin
    if Digit >= 10 then
        Conv := chr ( Digit + 55 )
    else
        Conv := chr ( Digit + 48 );
    end;
end;

( Convert integer to Hexadecimal ASCII string )
function Int_To_Hex ( Int : integer ) : Control_String_Type;
var
    D1, D2, D3, D4 : integer;
begin
    D4 := Int shr 12;
    Int := Int - D4 shl 12;
    D3 := Int shr 8;
    Int := Int - D3 shl 8;
    D2 := Int shr 4;
    D1 := Int - D2 shl 4;

    Int_To_Hex := Conv ( D4 ) + Conv ( D3 ) + Conv ( D2 ) + Conv ( D1 );
end;

```

```

-----
Name:      Input_Driver
Location:  INPUT.PAS
Purpose:   Input drivers for map-drawing program

Copyright (c) 1986, 1987 by
Doug George, 66 Shetland Way,
Kanata, Ontario, K2M 1S4
-----

{ Sound a beep }
procedure Beep;
begin
  Sound ( 440 );
  Delay ( 150 );
  NoSound;
end;

{ Output the prompt }
procedure Write_Prompt (
  Prompt : String_Type;
  Prompt_Data : PPrompt_Data_Type );

begin
  GotoXY ( 1, 22 );
  ClrEOL;
  write ( Prompt );
  GotoXY ( 1, 23 );
  ClrEOL;
  write ( '[ ' );

  with Prompt_Data do
    case Prompt_Mode of
      Boolean_Prompt :
        write ( 'Y / N ] > ' );
      Integer_Prompt :
        write ( Minimum_Integer : 1, ' - ',
                  Maximum_Integer : 1, ' ] > ' );
      String_Prompt :
        write ( Max_Length : 1, ' ] > ' );
      Real_Prompt :
        write ( Minimum_Real : 2 : 2, ' - ',
                  Maximum_Real : 2 : 2, ' ] > ' );
    end;
end;

{ Input a string }
procedure Get_String (
  Prompt      : String_Type;
  Length      : integer;
  var Data    : String_Type );

var
  Prompt_Data : PPrompt_Data_Type;
  OK : boolean;

begin
  Prompt_Data.Prompt_Mode := String_Prompt;
  Prompt_Data.Max_Length := Length;

  repeat
    Data := '';
    Write_Prompt ( Prompt, Prompt_Data );
    {$I-}
    readln ( Data );
    {$I+}

    OK := IOResult = 0;
  until OK;
end;

```

```

        if not OK then
            Beep;
until OK;
end;

{ Input a boolean }
procedure Get_Boolean (
    Prompt : String_Type;
    var Data : boolean );

var
    Prompt_Data : Prompt_Data_Type;
    Temp : char;
    OK : boolean;

begin
    Prompt_Data.Prompt_Mode := Boolean_Prompt;
repeat
    Temp := ' ';
    Write_Prompt ( Prompt, Prompt_Data );
    ($I-)
    readln ( Temp );
    ($I+)

    OK := (IOResult = 0) and
        ((Temp = 'Y') or
         (Temp = 'y') or
         (Temp = 'N') or
         (Temp = 'n'));
    if not OK then
        Beep;
until OK;

    Data := (Temp = 'Y') or (Temp = 'y');
end;

{ Input an integer number }
procedure Get_Integer (
    Prompt : String_Type;
    Minimum : integer;
    Maximum : integer;
    var Data : integer );

var
    Prompt_Data : Prompt_Data_Type;
    OK : boolean;

begin
    Prompt_Data.Prompt_Mode := Integer_Prompt;
    Prompt_Data.Minimum_Integer := Minimum;
    Prompt_Data.Maximum_Integer := Maximum;
repeat
    Data := MaxInt;
    Write_Prompt ( Prompt, Prompt_Data );
    ($I-)
    readln ( Data );
    ($I+)

    OK := IOResult = 0;
    if not OK then
        Beep;
    OK := OK and
        (Data <= Maximum) and
        (Data >= Minimum);
    if not OK and (Data <> MaxInt) then
        Beep;
until OK;

```

```

end;

( Input a real number )
procedure Get_Real (
    Prompt : String_Type;
    Minimum : real;
    Maximum : real;
    var Data : real );

var
    Prompt_Data : Prompt_Data_Type;
    OK : boolean;

begin
    Prompt_Data.Prompt_Mode := Real_Prompt;
    Prompt_Data.Minimum_Real := Minimum;
    Prompt_Data.Maximum_Real := Maximum;

repeat
    Data := 0;
    Write_Prompt ( Prompt, Prompt_Data );
    ($I-)
    readln ( Data );
    ($I+)

    OK := (IOResult = 0) and
        (Data <= Maximum) and
        (Data >= Minimum);
    if not OK then
        Beep;
until OK;
end;

( Get input using a menu )
procedure Get_Menu (
    Menu : Menu_Type;
    var Result : integer );

const
    Title_Left = 14;
    Title_Top = 5;
    Left_Margin = 10;
    Top_Margin = 7;

var
    I : integer;

begin
    clrscr;

    GotoXY ( Title_Left, Title_Top );
    write ( Menu.Title );

    for I := 1 to Menu.Max do
        begin
            GotoXY ( Left_Margin, Top_Margin + I );
            write ( I : 1, ' ', Menu.Option( I ) );
        end;

    Get_Integer ( 'Select Option', 1, Menu.Max, Result );
    Result := Result;
end;

( Get time from system )
procedure Get_DOS_Time (
    var Time_In : Time_Type;
    var Time_R : real );

var
    Reus : Registers;
    AH, AL, CH, CL, DH : byte;

```

```

    Hour_R, Min_R, Sec_R : integer;
    Hour, Min, Sec : String [ 2 ];

begin
with Regs do
begin
    AH := $2c;
    AX := AH shl 8 + AL;

    intr ( $21, Regs );

    Hour_R := CX shr 8;
    Min_R := CX mod 256;
    Sec_R := DX shr 8;

    Time_R := (Hour_R * 15.0 + Min_R / 4.0 + Sec_R / 240.0) * Deg To Rad;

    str ( Hour_R, Hour );
    str ( Min_R, Min );
    str ( Sec_R, Sec );

    if Hour_R <= 9 then
        Time_In := '0' + Hour + ':'
    else
        Time_In := Hour + ':'
    if Min_R <= 9 then
        Time_In := Time_In + '0';
    Time_In := Time_In + Min + ':'
    if Sec_R <= 9 then
        Time_In := Time_In + '0';
    Time_In := Time_In + Sec;
    end;
end;

( Get date from system )
procedure Get_DOS_Date (
    var Date_In : Date_Type );

var
    Regs      : Registers;
    Month, Day : String[ 2 ];
    Year      : String[ 4 ];
    DX, CX    : integer;

begin
with Regs do
begin
    AX := $2a shl 8;
    MsDos ( Regs );
    str ( CX, Year );
    str ( DX mod 256, Day );
    str ( DX shr 8, Month );
    if length ( Day ) < 2 then
        Day := '0' + Day;
    if length ( Month ) < 2 then
        Month := '0' + Month;
    Date_In := Year + '/' + Month + '/' + Day;
    end;
end;

procedure Get_Input_Time (
    Prompt : String_Type;
    var Time_In : Time_Type;
    var Time_R : real);

var
    OK      : boolean;
    Hour    : String[ 2 ];
    Minute  : String[ 2 ];
    Second  : String[ 2 ];
    Temp    : real;
    Result  : integer;

```



```

begin
repeat
  clrscr;
  gotoxy ( 7, 21 );
  write ( Prompt );
  gotoxy ( 7, 22 );
  write ( 'HH:MM:SS' );
  gotoxy ( 7, 23 );
  ($I-)
  read ( Time_in );
  ($I+);

  OK := (IOResult = 0);
  while Length ( Time_in ) < 8 do
    Time_in := Time_in + '0';

  if OK then
    begin
      Hour := Copy ( Time_in, 1, 2 );
      Minute := Copy ( Time_in, 4, 2 );
      Second := Copy ( Time_in, 7, 2 );

      Val ( Hour, Temp, Result );
      OK := (Result = 0) and (Temp >= 0.0) and (Temp < 24.0);
      Time_R := Temp;
      end;

    if OK then
      begin
        Val ( Minute, Temp, Result );
        OK := (Result = 0) and (Temp >= 0.0) and (Temp < 60.0);
        Time_R := Time_R + Temp / 60.0;
        end;

    if OK then
      begin
        Val ( Second, Temp, Result );
        OK := (Result = 0) and (Temp >= 0.0) and (Temp < 60.0);
        Time_R := Time_R + Temp / 3600.0;
        end;

    OK := OK and (Time_R < 360.0);

    if not OK then
      Beep;

until OK;

Time_R := Time_R * Hour_To_Rad;
Time_in [ 3 ] := ':';
Time_in [ 6 ] := ':';

end;

procedure Get_Input_Dec (
  Prompt : String_Type;
  var Dec_R : real );

var
  OK      : boolean;
  Sign    : String[ 1 ];
  Degree  : String[ 2 ];
  Minute  : String[ 2 ];
  Second  : String[ 2 ];
  Dec_in  : String[ 9 ];
  Temp    : real;
  Result  : integer;

begin
repeat

```

```

ClrScr;
GotoXY ( 7, 21 );
write ( Prompt );
GotoXY ( 7, 22 );
write ( 'SDD:MM:SS' );
GotoXY ( 7, 23 );
{$I-}
read ( Dec_In );
{$I+}

OK := (IOResult = 0);
while Length ( Dec_In ) < 9 do
  Dec_In := Dec_In + '0';

if OK then
begin
  Sign := Copy ( Dec_In, 1, 1 );
  Degree := Copy ( Dec_In, 2, 2 );
  Minute := Copy ( Dec_In, 5, 2 );
  Second := Copy ( Dec_In, 8, 2 );

  Val ( Degree, Temp, Result );
  OK := (Result = 0) and (Temp >= 0.0) and (Temp <= 90.0);
  Dec_R := Temp;
end;

if OK then
begin
  Val ( Minute, Temp, Result );
  OK := (Result = 0) and (Temp >= 0.0) and (Temp < 60.0);
  Dec_R := Dec_R + Temp / 60.0;
end;

if OK then
begin
  Val ( Second, Temp, Result );
  OK := (Result = 0) and (Temp >= 0.0) and (Temp < 60.0);
  Dec_R := Dec_R + Temp / 3600.0;
end;

OK := OK and (Dec_R <= 90.0);
if Sign = '-' then
  Dec_R := -Dec_R;

if not OK then
  Beep;

until OK;

Dec_R := Dec_R * Deg_To_Rad;

end;

procedure Get_Input_Date (
  var Date_In : Date_Type );

var
  OK      : boolean;
  Year   : String [ 4 ];
  Month  : String [ 2 ];
  Day    : String [ 2 ];
  Temp   : real;
  Result : integer;

begin
  repeat
    ClrScr;
    GotoXY ( 1, 23 );
    writeln ( '          YYYY/MM/DD' );
    write   ( 'New Date   = ' );
    {$I-}
    read ( Date_In );
  until

```

```

(91+)

OK := (IOResult = 0);
OK := OK and (Length ( Date_In ) = 10);

if OK then
begin
    Year := Copy ( Date_In, 1, 4 );
    Month := Copy ( Date_In, 6, 2 );
    Day := Copy ( Date_In, 9, 2 );

    Val ( Year, Temp, Result );
    OK := (Result = 0) and (Temp >= 0.0) and (Temp <= 9999.0);
end;

if OK then
begin
    Val ( Month, Temp, Result );
    OK := (Result = 0) and (Temp >= 0.0) and (Temp <= 12.0);
end;

if OK then
begin
    Val ( Day, Temp, Result );
    OK := (Result = 0) and (Temp >= 0.0) and (Temp <= 31.0);
end;

if not OK then
    Beep;

until OK;

Date_In [ 5 ] := '/';
Date_In [ 8 ] := '/';

end;

(* Initialize the Date and Time *)
procedure Date_Time ( T_Mode : integer );

begin
    case T_Mode of
        (* Present Time *)
        1 : begin
            Get_Dos_Date ( Date_String );
            Get_Dos_Time ( Time_String, Time );
            Calculate_Siderial ( Date_String, Time, Time_Zone,
                                Longitude, Siderial );
            Time_To_String ( Siderial, Siderial_String );
            end;

        (* Manual *)
        2 : begin
            Get_Input_Date ( Date_String );
            Get_Input_Time ( 'Enter Standard Time', Time_String, Time );
            Calculate_Siderial ( Date_String, Time, Time_Zone,
                                Longitude, Siderial );
            Time_To_String ( Siderial, Siderial_String );
            ClrScr;
            end;

        (* Manual Siderial *)
        3 : begin
            Date_String := '0000/00/00';
            Time_String := '00:00:00';
            Time := 0;
            Get_Input_Time ( 'Enter Siderial Time', Siderial_String, Siderial );
            ClrScr;
            end;

        (* No Change *)
        4 : begin

```

```
        ClrScr;  
    end;  
end;  
end;
```

```

-----
Name:      Read_File
Location:  READFILE.PAS
Purpose:   Reads in a star database from disk

Copyright (c) 1986, 1987 by
Doug George, 66 Shetland Way
Kanata, Ontario, K2M 1S4
-----
}

type
  File_String = String[ 30 ];

{ Load a file into a given page }
procedure Load_File (
  Page      : integer;
  File_Name : File_String );

var
  Star_File : file of Star_Array_Type;
  I : integer;

begin
  assign ( Star_File, File_Name );
  reset  ( Star_File );
  read   ( Star_File, Star_Table[ Page ].Star );

  close ( Star_File );

  Max_Stars := I;

  repeat
    I := I - 1;
  until (I <= 0) or (Star_Table[ Page ].Star[ I ].Mag < MaxMag);

  Star_Table[ Page ].First_Star := I;
  Star_Table[ Page ].Last_Star  := I;
  Star_Table[ Page ].Number_Of_Stars := I;
end;

{ Scan for the first and last declination stars }
procedure Scan (
  Page      : integer;
  First_Dec : real;
  Last_Dec  : real );

var
  I : integer;

begin
  with Star_Table[ Page ] do
    begin
      I := 2;
      while (I <= Number_Of_Stars) and
        (Star[ I ].Dec < First_Dec) do
        I := I + 1;
      First_Star := I - 1;

      while (I <= Number_Of_Stars) and
        (Star[ I ].Dec < Last_Dec) do
        I := I + 1;
      Last_Star := I - 1;
    end;
  end;
end;

```

```

end;

( Globals for use by procedures below )
var
  Start_RA      : real;
  End_RA        : real;
  Start_Dec     : real;
  End_Dec       : real;
  Dest_Page     : integer;

( Copy stars between pages )
procedure Copy_Stars;

var
  Source        : integer;
  Dest          : integer;
  OK_To_Copy    : boolean;

begin
  Dest := Star_Table[ Dest_Page ]^.Number_Of_Stars + 1;
  Source := Star_Table[ Max_Pages ]^.First_Star;

  ( Copy the stars )
  while (Dest_Page < Max_Pages) and
    (Source <= Star_Table[ Max_Pages ]^.Last_Star) and
    not keypressed do

    begin
      ( If the destination page is not full, copy over stars in range )
      if Dest < Max_Stars then
        begin
          ( Determine whether RA within range, accounting for modulo 2 PI )
          if Start_RA < End_RA then
            OK_To_Copy :=
              (Start_RA < Star_Table[ Max_Pages ]^.Star[ Source ].RA) and
              (End_RA > Star_Table[ Max_Pages ]^.Star[ Source ].RA)
          else
            OK_To_Copy :=
              (Start_RA < Star_Table[ Max_Pages ]^.Star[ Source ].RA) or
              (End_RA > Star_Table[ Max_Pages ]^.Star[ Source ].RA);

          if OK_To_Copy then
            begin
              Star_Table[ Dest_Page ]^.Star[ Dest ] :=
                Star_Table[ Max_Pages ]^.Star[ Source ];

              ( Convert the star's position to X-Y coordinates )
              Project_Point ( Star_Table[ Dest_Page ]^.Star[ Dest ].RA,
                             Star_Table[ Dest_Page ]^.Star[ Dest ].Dec );
              Dest := Dest + 1;
            end;
          end;
          Source := Source + 1;

          ( If we are out of space, then set up next page )
          if Dest > Max_Stars then
            begin
              ( Set parameters of destination table )
              Star_Table[ Dest_Page ]^.Number_Of_Stars := Dest - 1;
              Star_Table[ Dest_Page ]^.Last_Star := Dest - 1;

              ( Increment to next page )
              Dest_Page := Dest_Page + 1;
              Dest := 1;
            end;
          end;
        end;

      Star_Table[ Dest_Page ]^.Number_Of_Stars := Dest - 1;
      Star_Table[ Dest_Page ]^.Last_Star := Dest - 1;
    end;
  end;

```

```

( Increment to next page for next iteration, if required )
if Dest > (Max_Stars + 1) then
  begin
    Dest_Page := Dest_Page + 1;
    Dest := 1;
  end;

( Clear out source table )
Star_Table[ Max_Pages ]^.Number_Of_Stars := 0;
Star_Table[ Max_Pages ]^.First_Star := 1;
Star_Table[ Max_Pages ]^.Last_Star := 0;

end;

( Load buffers with a particular file )
procedure Load_Buffers (
  File_Name   : File_String;
  File_Number : integer );

var
  Temp : File_String;

begin

  Str ( File_Number, Temp );
  if File_Number < 100 then
    Temp := '0' + Temp;
  if File_Number < 10 then
    Temp := '0' + Temp;

  Temp := File_Name + Temp;
  Load_File ( Max_Pages, Temp );
  Scan ( Max_Pages, Start_Dec, End_Dec );
  Copy_Stars;

end;

( Read in the necessary files )
procedure Read_File;

( Local parameters )
var
  Page       : integer;
  File_Number : integer;
  Start_File : integer;
  End_File   : integer;
  Range      : real;

begin

  ( Clear out previous data )
  for Page := 1 to Max_Pages do
    begin
      Star_Table[ Page ]^.Number_Of_Stars := 0;
      Star_Table[ Page ]^.Last_Star := 0;
      Star_Table[ Page ]^.First_Star := 1;
    end;

  ( Set initial page )
  Dest_Page := 1;

  ( Calculate range )
  if Initial_Data.Alt_Az_Mode then
    begin
      Start_RA := 0;
      End_RA := 2 * PI;
      Start_Dec := -PI;
      End_Dec := PI;
    end

```

```

else if Initial_Data.Scale >= (90 * Deg_To_Rad) then
  begin
    Range := Initial_Data.Scale * 0.5;
    Start_RA := Initial_Data.Right_Ascension - Range;
    End_RA := Initial_Data.Right_Ascension + Range;
    Start_Dec := -PI;
    End_Dec := PI;
  end
else
  begin
    Range := Initial_Data.Scale;
    if Range < 0.04 then
      Range := 0.04;
    Start_RA := Initial_Data.Right_Ascension - Range * 2;
    End_RA := Initial_Data.Right_Ascension + Range * 2;
    Start_Dec := Initial_Data.Declination - Range;
    End_Dec := Initial_Data.Declination + Range;
  end;

  { Ensure within range }
  if Start_RA < 0 then
    Start_RA := Start_RA + Two_PI;
  if End_RA > Two_PI then
    End_RA := End_RA - Two_PI;
  if Start_Dec < -Half_PI then
    Start_Dec := -Half_PI;
  if End_Dec > Half_PI then
    End_Dec := Half_PI;

  { Find range of SAO and load, if applicable }
  if Initial_Data.Scale <= (6 * Deg_To_Rad) then
    begin
      Start_File := 0;
      End_File := Max_SAO_Files - 1;
      for File_Number := 0 to Max_SAO_Files - 1 do
        begin
          if (SAO_RA_Table[ File_Number ] <= Start_RA) and
             (SAO_RA_Table[ File_Number + 1 ] >= Start_RA) then
            Start_File := File_Number;
          if (SAO_RA_Table[ File_Number ] <= End_RA) and
             (SAO_RA_Table[ File_Number + 1 ] >= End_RA) then
            End_File := File_Number;
          end;
        File_Number := Start_File - 1;
        if not keypressed then
          repeat
            File_Number := (File_Number + 1) mod Max_SAO_Files;
            Load_Buffers ( '\STARDATA\SAODATA.', File_Number );
          until (File_Number = End_File) or keypressed;
        end;
      end;

  { Load bright stars }
  if Initial_Data.Scale <= (50 * Deg_To_Rad) then
    begin
      Start_File := 0;
      End_File := Max_YBSC_Files - 1;
      for File_Number := 0 to Max_YBSC_Files - 1 do
        begin
          if (YBSC_RA_Table[ File_Number ] <= Start_RA) and
             (YBSC_RA_Table[ File_Number + 1 ] >= Start_RA) then
            Start_File := File_Number;
          if (YBSC_RA_Table[ File_Number ] <= End_RA) and
             (YBSC_RA_Table[ File_Number + 1 ] >= End_RA) then
            End_File := File_Number;
          end;
        File_Number := Start_File - 1;
        if not keypressed then
          repeat
            File_Number := (File_Number + 1) mod Max_YBSC_Files;
            Load_Buffers ( '\STARDATA\YBSCDATA.', File_Number );
          until (File_Number = End_File) or keypressed;
        end;
      end;

  else if not keypressed then { Load Brightest Stars only }

```



```

begin
  Load_File ( Max_Pages, '\STARDATA\STARS.BIN' );
  Copy_Stars;
end;

end;

procedure Load_Temp_Buffer (
  File_Name : File_String;
  File_Number : integer );

var
  Temp : File_String;

begin
  str ( File_Number, Temp );
  if File_Number < 100 then
    Temp := '0' + Temp;
  if File_Number < 10 then
    Temp := '0' + Temp;

  Temp := File_Name + Temp;
  Load_File ( Max_Pages, Temp );

end;

(* Identify a centered object *)
procedure Identify_Object (
  RA : real;
  Dec : real;
  var Name : Name_Type );

var
  I : integer;
  J : integer;
  Dist : real;
  Min_Dist : real;
  Min_Dist_Mag : integer;

begin
  (* Indicate none found yet *)
  Min_Dist := MaxInt;
  Min_Dist_Mag := MaxInt;

  (* Load named stars *)
  Load_File ( Max_Pages, '\STARDATA\STARS.BIN' );

  I := 1;
  while not keypressed and (I <= Max_Pages) do
    begin
      J := Star_Table[ I ].First_Star;
      while not keypressed and (J <= Star_Table[ I ].Last_Star) do
        begin
          if I = Max_Pages then
            Project_Point ( Star_Table[ I ].Star[ J ].RA,
                          Star_Table[ I ].Star[ J ].Dec );

          Dist := abs ( Star_Table[ I ].Star[ J ].RA -
                        Initial_Data.X_Offset - X_Center ) +
                    abs ( Star_Table[ I ].Star[ J ].Dec -
                        Initial_Data.Y_Offset - Y_Center );

          if (Dist <= 10.0) and
             (Star_Table[ I ].Star[ J ].Mag < Min_Dist_Mag) then
            begin
              Min_Dist := Dist;
              Min_Dist_Mag := Star_Table[ I ].Star[ J ].Mag;
              Name := Star_Table[ I ].Star[ J ].Name;
            end;
          J := J + 1;
        end;
      I := I + 1;
    end;
  end;

```

```

        J := J + 1;
    end;

    I := I + 1;
end;

if Min_Dist > 10 then
    Name := 'NONE'
else if Name = ' ' then
    Name := 'NONAME';

{ Erase last page }
Star_Table[ Max_Pages ]^.Number_Of_Stars := 0;
Star_Table[ Max_Pages ]^.First_Star := 1;
Star_Table[ Max_Pages ]^.Last_Star := 0;

end;

{ Locate an object }
procedure Locate_Object (
    var Name : Name_Type );

var
    I : integer;
    J : integer;
    Found : boolean;
    Non_SAO : boolean;
    RA_Temp : real;
    Dec_Temp : real;
    Temporary : Projection_Type;

begin
    { Decide whether we are looking for an SAO or named star. }
    { and change to upper case if required. }
    Non_SAO := false;
    for I := 1 to Length ( Name ) do
        begin
            Non_SAO := Non_SAO or not (Name[ I ] in [ '0' .. '9' ]);
            if Name [ I ] in [ 'a' .. 'z' ] then
                Name [ I ] := chr ( ord ( Name [ I ] ) - $20 );
            end;
        end;

    { If named star, we need to look in small database }
    if Non_SAO then
        begin
            Load_File ( Max_Pages, '\STARDATA\STARS.BIN' );
            I := Max_Pages;
        end
    else { we need to look in large database }
        begin
            Temporary := Initial_Data.Projection;
            Initial_Data.Projection := No_Projection;
            Read_File;
            I := 1;
        end;

    { Search for the star }
    Found := false;
    while (I <= Max_Pages) and not Found do
        begin
            J := Star_Table[ I ]^.First_Star;
            while (J <= Star_Table[ I ]^.Last_Star) and not Found do
                begin
                    Found := Name = Star_Table[ I ]^.Star[ J ].Name;
                    if not Found then
                        J := J + 1;
                    end;
                end;
            if not Found then
                I := I + 1;
            end;
        end;
end;

```

```

( Adjust to center of screen )
if Found then
  begin
    ( Get new RA and Dec )
    Initial_Data.Right_Ascension := Star_Table[ I ]^.Star[ J ].RA;
    Initial_Data.Declination := Star_Table[ I ]^.Star[ J ].Dec;
  end
else
  Name := 'NONE ';

( For newly-loaded data, go through and re-project )
if not Non_SAO then
  begin
    Initial_Data.Projection := Temporary;
    Pre_Calculate;
    for I := 1 to Max_Pages - 1 do
      for J := Star_Table[ I ]^.First_Star to
        Star_Table[ I ]^.Last_Star do
        Project_Point ( Star_Table[ I ]^.Star[ J ].RA,
          Star_Table[ I ]^.Star[ J ].Dec );
      end
    end
  else
    begin
      ( If we searched through the small database, a re-load is required )
      Pre_Calculate;
      Read_File;
      end;

    ( Clear out temporary area )
    Star_Table[ Max_Pages ]^.Number_Of_Stars := 0;
    Star_Table[ Max_Pages ]^.First_Star := 1;
    Star_Table[ Max_Pages ]^.Last_Star := 0;

  end;

```

```

-----
Name:      Initialization
Location:  INITIAL.PAS
Purpose:   Initialization routines for map-drawing program

Copyright (c) 1986, 1987 by
Doug George, 66 Shetland Way
Kanata, Ontario, K2M 1S4
-----

```

```

( This routine initializes the program )
procedure Initialize;

var
  I : integer;
  SAO_RA_File : file of SAO_RA_Table_Type;
  YBSC_RA_File : file of YBSC_RA_Table_Type;

begin
  ( Set up conversion constants )
  Deg_To_Rad := PI / 180.0;      ( Multiply to get radians )
  Rad_To_Deg := 1.0 / Deg_To_Rad; ( Multiply to get degrees )
  Hour_To_Rad := PI / 12.0;     ( Multiply to get hours )
  Rad_To_Hour := 1.0 / Hour_To_Rad; ( Multiply to get radians )
  Two_Pi := 2.0 * Pi;
  Half_Pi := Pi / 2.0;
  Quarter_Pi := Pi / 4.0;

  ( Store site variables )

  Time_Zone := 5;
  Longitude := 75.0 * Deg_To_Rad;
  Latitude := 45.0 * Deg_To_Rad;
  Map_Data.Sin_Latitude := sin ( Latitude );
  Map_Data.Cos_Latitude := cos ( Latitude );
  Telescope_Scale := Default_Telescope_Scale * Deg_To_Rad;

  ( These are determined from the display driver )
  X_Scale := Initial_X_Scale;
  Y_Scale := Initial_Y_Scale;

  Magnitude_Offset := 0;
  Siderial := 90;
  Initial_Data.X_Offset := 0;
  Initial_Data.Y_Offset := 0;

  ( Load RA tables )
  assign ( SAO_RA_File, '\stardata\sacra.bin' );
  reset ( SAO_RA_File );
  read ( SAO_RA_File, SAO_RA_Table );
  assign ( YBSC_RA_File, '\stardata\yosra.bin' );
  reset ( YBSC_RA_File );
  read ( YBSC_RA_File, YBSC_RA_Table );

  ( Clear star files )
  for I := 1 to Max_Pages do
    begin
      new ( Star_Table[ I ] );
      Star_Table[ I ].First_Star := 1;
      Star_Table[ I ].Last_Star := 0;
      Star_Table[ I ].Number_Of_Stars := 0;
    end;

  ( Load in graphics font )
  Initialize_Font;
end;

```

```

-----
Name:      Draw_Stars
Location:  DRAW.PAS
Purpose:   Displays stars on the screen

Copyright (c) 1986, 1987 by
Doug George, 66 Shetland Way
Kanata, Ontario, K2M 1S4
-----

( Calculate map projection parameters )
procedure Set_Up (
    Right_Ascension : real; ( Azimuth )
    Declination      : real; ( Altitude )
    Alt_Az_Mode      : boolean;
    Scale_Factor     : real ); ( Degrees )

begin

Magnitude_Offset := 0;
if Alt_Az_Mode then
    Initial_Data.Projection := Lambert_Conformal
else if Scale_Factor >= (90.0 * Deg_To_Rad) then
    Initial_Data.Projection := Athmuthal_Stereographic
else
    begin
        if Scale_Factor <= (1 * Deg_To_Rad) then
            Magnitude_Offset := 500;
        else if Scale_Factor <= (6.0 * Deg_To_Rad) then
            Magnitude_Offset := 300;
        if abs ( Declination ) < Scale_Factor then
            Initial_Data.Projection := Cylindrical
        else
            Initial_Data.Projection := Lambert_Conformal;
        end;
    end;

Initial_Data.Right_Ascension := Right_Ascension;
Initial_Data.Declination := Declination;
Initial_Data.Alt_Az_Mode := Alt_Az_Mode;
Initial_Data.Scale := Scale_Factor;

end;

procedure Re_Set_Up;
begin
with Initial_Data do
    Set_Up ( Right_Ascension, Declination, Alt_Az_Mode, Scale );
end;

procedure Move_Position (
    Delta_RA : real;
    Delta_Dec : real );

begin
with Initial_Data do
begin
    Right_Ascension := Right_Ascension + Delta_RA;
    if Right_Ascension > Two_PI then
        Right_Ascension := Right_Ascension - Two_PI
    else if Right_Ascension < 0.0 then
        Right_Ascension := Right_Ascension + Two_PI;

    Declination := Declination + Delta_Dec;
    if Declination > Half_PI then
        Declination := Half_PI
    else if Declination < -Half_PI then
        Declination := -Half_PI;
end;
end;

```

```

X_Offset := Right_Ascension;
Y_Offset := Declination;
Project_Point ( X_Offset, Y_Offset );
X_Offset := X_Offset - X_Center;
Y_Offset := Y_Offset - Y_Center;
end;

end;

( Draw Star )
procedure Draw_Stars;

type
  Status_Type = ( Done, Recalculate, Redisplay, Continue );
  Adjust_Type = ( Normal, Adjust_Screen, Adjust_Scope );

( "Permanent" variables )
var
  Show_Position : boolean;
  Adjust : Adjust_Type;
  Display_On : boolean;
  Dec_Reverse : boolean;
  Slew_Distance : real;
  Name : Name_Type;
  Delta_RA : real;
  Delta_Dec : real;

( local procedure )
  procedure Calculating;
  begin
    { Put up message }
    Clear_Screen;
    Toggle_Banks;
    Write_Text ( 'CALCULATING', 208, 300 );
    Toggle_Banks;

    end;

( local procedure )
  procedure Display_Position;

  var
    RA_String : Time_Type;
    Dec_String : Dec_Type;

  begin
    Time_To_String ( Initial_Data.Right_Ascension, RA_String );
    Dec_To_String ( Initial_Data.Declination, Dec_String );
    Write_Text ( RA_String, 135, 0 );
    Write_Text ( Dec_String, 315, 0 );
    end;

( local procedure )
  procedure Remove_Backlash;

  begin
    write ( aux, 'M0000000000' );
    end;

( local procedure )
  procedure Move_Scope (
    RA_Distance : real;
    Dec_Distance : real );

  begin
    RA_Distance := -RA_Distance * RA_Steps_Per_Radian;
    Dec_Distance := -Dec_Distance * Dec_Steps_Per_Radian;

```

```

( If the declination motor reverse has been set, change sign ;
if Dec_Reverse then
  Dec_Distance := -Dec_Distance;

if Initial_Data.Scale = Telescope_Scale then
begin
  if RA_Distance <> 0 then
    write ( aux, 'R' + Int_To_Hex ( Round ( RA_Distance ) ) );

    if Dec_Distance <> 0 then
      write ( aux, 'D' + Int_To_Hex ( Round ( Dec_Distance ) ) )
    end
  else
begin
  Delta_RA := Delta_RA + RA_Distance;
  Delta_Dec := Delta_Dec + Dec_Distance;
end;
end;

( local procedure )
procedure Slew_Scope;

var
  RA_Big : integer;
  Dec_Big : integer;
  RA_Small : integer;
  Dec_Small : integer;

begin
  if Adjust = Adjust_Screen then
begin
  Delta_RA := 0.0;
  Delta_Dec := 0.0;
end
else if (Delta_RA <> 0.0) or (Delta_Dec <> 0.0) then
begin
  Delta_RA := Delta_RA;
  Delta_Dec := Delta_Dec;

  ( Convert Delta_RA to 20-bit Hex )
  RA_Big := Trunc ( Delta_RA / 16.0 );
  RA_Small := Round ( abs ( Delta_RA - RA_Big * 16.0 ) );
  if (Delta_RA < 0) and (RA_Small > 0) then
begin
  RA_Big := RA_Big - 1;
  RA_Small := 16 - RA_Small;
end;

  if Dec_Reverse then
    Delta_Dec := -Delta_Dec;

  ( Convert Delta_Dec to 20-bit Hex )
  Dec_Big := Trunc ( Delta_Dec / 16.0 );
  Dec_Small := Round ( abs ( Delta_Dec - Dec_Big * 16.0 ) );
  if (Delta_Dec < 0) and (Dec_Small > 0) then
begin
  Dec_Big := Dec_Big - 1;
  Dec_Small := 16 - Dec_Small;
end;

  ( write out high speed slew command )
  write ( aux, 'M' + Int_To_Hex ( RA_Big ) + Conv ( RA_Small )
    + Int_To_Hex ( Dec_Big ) + Conv ( Dec_Small ) );

  ( Reset Delta's to zero )
  Delta_RA := 0.0;
  Delta_Dec := 0.0;
end;
end;

( local procedure )
procedure Process_Key (

```

```

var Status      : Status_Type );

var
  Status_Flag : boolean;
  Key : Char;
  Position : Time_Type;
  R : real;
  D : real;
  Data : String_Type;

begin
  ( Retain Recalculate status )
  Status_Flag := Status = Recalculate;

  ( Get key from keyboard and process )
  read ( kbd, Key );
  if ord ( Key ) = 27 then
    read ( kbd, Key )
  else if ord ( Key ) = 13 then
    Status := Done
  else if Key <> ' ' then
    Key := chr( 0 );

    case Key of
      'G', 'H', 'I', 'K', 'M', 'O', 'P', 'Q' :
        begin
          R := 0;
          D := 0;
          case Key of
            'G', 'H', 'I' : D := Slew_Distance;
            'O', 'P', 'Q' : D := -Slew_Distance;
          end;
          case Key of
            'G', 'K', 'O' : R := Slew_Distance;
            'I', 'M', 'Q' : R := -Slew_Distance;
          end;
          if Adjust <> Adjust_Screen then
            Move_Scope ( R, D );
          if Adjust <> Adjust_Scope then
            Move_Position ( R, D );
          Status := Redisplay;
        end;
      ';' : ( F1 )
        begin
          Status := Recalculate;
          Initial_Data.Scale := Telescope_Scale;
          Slew_Distance := Initial_Data.Scale * 0.2;
          Slew_Scope;
        end;
      '=' : ( F3 )
        begin
          Status := Recalculate;
          Initial_Data.Scale := 2.0 * Deg_To_Rad;
          Slew_Distance := Initial_Data.Scale * 0.2;
        end;
      '?' : ( F5 )
        begin
          Status := Recalculate;
          Initial_Data.Scale := 5.0 * Deg_To_Rad;
          Slew_Distance := Initial_Data.Scale * 0.2;
        end;
      'A' : ( F7 )
        begin
          Status := Recalculate;
          Initial_Data.Scale := 30.0 * Deg_To_Rad;
          Slew_Distance := Initial_Data.Scale * 0.2;
        end;
      'C' : ( F9 )
        begin
          Status := Recalculate;
          Initial_Data.Scale := 90.0 * Deg_To_Rad;
          Slew_Distance := Initial_Data.Scale * 0.2;
        end;
      '^' : ( CTRL-F1 )

```



```

Slew_Distance := 0.01 * Initial_Data.Scale;
'' : ( CTRL-F3 )
Slew_Distance := 0.02 * Initial_Data.Scale;
'b' : ( CTRL-F5 )
Slew_Distance := 0.05 * Initial_Data.Scale;
'd' : ( CTRL-F7 )
Slew_Distance := 0.10 * Initial_Data.Scale;
'f' : ( CTRL-F9 )
Slew_Distance := 0.20 * Initial_Data.Scale;
'' : ( F2 )
begin
  ( Get new RA and Dec )
  Set_Text_Mode;
  R := Initial_Data.Right_Ascension;
  D := Initial_Data.Declination;
  Get_Input_Time ( 'Right Ascension', Position,
                  Initial_Data.Right_Ascension );
  Get_Input_Dec ( 'Declination', Initial_Data.Declination );
  Set_Graphics_Mode;

  ( Slew motors if required )
  if Adjust <> Adjust_Screen then
    begin
      R := Initial_Data.Right_Ascension - R;
      if R > PI then
        R := R - Two_PI;
      else if R < -PI then
        R := R + Two_PI;
      D := Initial_Data.Declination - D;

      ( Move telescope )
      if Initial_Data.Scale = Telescope_Scale then
        begin
          ( Force immediate high-speed slew )
          Initial_Data.Scale := 2.0 * Deg_To_Rad;
          Move_Scope ( R, D );
          Initial_Data.Scale := Telescope_Scale;
          Slew_Scope;
        end
      else
        Move_Scope ( R, D );
      end;

      Status := Recalculate;
    end;
  '' : ( F4 )
  begin
    Display_On := not Display_On;
    Status := Redisplay;
  end;
  'a' : ( F6 )
  begin
    case Adjust of
      Normal      : Adjust := Adjust_Screen;
      Adjust_Screen : Adjust := Adjust_Scope;
      Adjust_Scope : Adjust := Normal;
    end;
    Status := Redisplay;
  end;
  '3' : ( F8 )
  begin
    Show_Position := not Show_Position;
    Status := Redisplay;
  end;
  '0' : ( F10 )
  Status := Recalculate;
  '' : ( CTRL-F2 )
  begin
    Set_Text_Mode;
    GotoXY ( 7, 10 );
    writeln ( 'Present Scale = ', Telescope_Scale * Rad_To_Deg:5:2 );
    Get_Real ( 'Enter Scale', 0.1, 80.0, Initial_Data.Scale );
    Initial_Data.Scale := Initial_Data.Scale * Deg_To_Rad;

```

```

Telescope_Scale := Initial_Data.Scale;
Slew_Distance := Initial_Data.Scale * 0.2;
Status := Recalculate;
Set_Graphics_Mode;
Slew_Scope;
end;
'a' : ( CTRL-F4 )
Dec_Reverse := not Dec_Reverse;
'c' : ( CTRL-F6 )
begin
Set_Text_Mode;
GotoXY (7, 10);
writeln ( 'Present X Scale = ', X_Scale:5:2 );
Get_Real ( 'Enter X_Scale', 1.0, 3.0, X_Scale );
GotoXY (7, 10);
ClrEol;
writeln ( 'Present Y Scale = ', Y_Scale:5:2 );
Get_Real ( 'Enter Y_Scale', 1.0, 3.0, Y_Scale );
Status := Recalculate;
Set_Graphics_Mode;
end;
'e' : ( CTRL-F8 )
begin
Calculating;
Identify_Object ( Initial_Data.Right_Ascension,
                  Initial_Data.Declination,
                  Name );
Status := Redisplay;
end;
'g' : ( CTRL-F10 )
begin
Set_Text_Mode;
Get_String ( 'ENTER NAME', Max_Name, Data );
Name := Data;
Set_Graphics_Mode;
Calculating;

( Locate Object and Slew )
R := Initial_Data.Right_Ascension;
D := Initial_Data.Declination;
Locate_Object ( Name );

( Slew motors if required )
if Adjust <> Adjust_Screen then
begin
R := Initial_Data.Right_Ascension - R;
if R > PI then
R := R - Two_PI;
else if R < -PI then
R := R + Two_PI;
D := Initial_Data.Declination - D;

( Move telescope )
if Initial_Data.Scale = Telescope_Scale then
begin
( Force immediate high-speed slew )
Initial_Data.Scale := 2.0 * Deg_To_Rad;
Move_Scope ( R, D );
Initial_Data.Scale := Telescope_Scale;
Slew_Scope;
end
else
Move_Scope ( R, D );
end;

Status := Redisplay;
end;
:
begin
if Initial_Data.Scale = Telescope_Scale then
Remove_Backlash;
else
Slew_Scope;
end;

```

```

        else;
        end;

    { Force Recalculation if entered with Recalculate status }
    if Status_Flag and (Status <> Done) then
        Status := Recalculate;
    end;

local procedure
    procedure Display_Stars;
    var
        Star          : integer;
        Star_Altitude : real;
        Star_Azimuth  : real;
        I             : integer;
        Cross         : boolean;
    begin
        { Loop from 1st page to Max_Pages - 1 ... Max_Pages used as buffer }
        I := 0;
        Cross := (Adjust <> Normal) and (Initial_Data.Scale = Telescope_Scale);
        repeat
            { Scan through pages }
            I := I + 1;
            { Output all stars on the page }
            Star := Star_Table[ I ].First_Star;
            while (Star <= Star_Table[ I ].Last_Star) and not keypressed do
                begin
                    with Star_Table[ I ].Star[ Star ] do
                        Display ( RA, Dec, Mag, Cross );
                    end;
                    Star := Star + 1;
                end;
            until keypressed or (I >= Max_Pages);

        end;

var
    Status : Status_Type;
    Clear  : boolean;

begin
    { Start telescope at zero }
    Remove Backlash;
    Delta RA := 0.0;
    Delta Dec := 0.0;

    Show_Position := true;
    Show_Distance := Initial_Data.Scale * 0.2;
    Clear := false;
    Adjust := Normal;
    Display_On := true;
    Dec_Reverse := false;
    Name := ' ';

    repeat
        { Set up map projection }
        Pre_Calculate;

        { Put up message }
        Calculating;

        { Get data from disk }
        Read_File;

        { Display stars }
        if keypressed then
            Status := Recalculate
        else
            Status := Redisplay;
    end;

```

```

repeat
  { Re-display stars if changed }
  if Status = Redisplay then
    begin
      { If the update screen has not been cleared, do so }
      if not (Clear or keypressed) then
        begin
          Clear_Screen;
          Clear := true;
        end;

      { Check for a key pressed during the clear operation }
      if not keypressed then
        Clear := false;

      { Show RA and Dec }
      if not keypressed and Show_Position then
        begin
          Display_Position;
          if (Adjust = Normal) or
            (Initial_Data.Scale <> Telescope_Scale) then
            Crosshairs ( X_Center, Y_Center, 0 );
        end;

      { Indicate if in adjustment mode }
      if not keypressed then
        if Adjust = Adjust_Scope then
          Write_Text ( 'ADJUST SCOPE', 200, 30 );
        else if Adjust = Adjust_Screen then
          Write_Text ( 'ADJUST SCREEN', 194, 30 );

      { Display star name }
      if not keypressed and (Name <> ' ') then
        Write_Text ( Name, X_Center + 16, Y_Center );
      Name := ' ';

      { Display stars }
      if not keypressed and Display_On then
        Display_Stars;

      { Change banks only if an entire screen was written }
      if not keypressed then
        begin
          Toggle_Banks;
          Status := Continue;
          Clear := false;
        end;
    end;

    { Read a key from the keyboard and process }
    Process_Key ( Status );

  until (Status = Done) or (Status = Recalculate);
  Re_Set_Up;
  until Status = Done;

  Stop Motion ;
  write ( aux, 'S' );

end;

{ Draw a coordinate grid }
procedure Draw_Grid;
var
  RA      : real;
  Dec     : real;
  I       : integer;
  J       : integer;
  Magnitude : integer;
  Alt     : real;

```

```

    Az      : real;

begin
  Pre_Calculator;
  Magnitude := 0;

  I := 0;
  repeat
    J := 9;
    repeat
      RA := Int ( I ) * 10. * Deg_To_Rad;
      Dec := Int ( J ) * 10. * Deg_To_Rad;

      Project_Point ( RA, Dec );
      Display ( RA, Dec, Magnitude, false );

      J := J - 1;
    until ( J < -9) or keypressed;

    I := I + 1;
  until ( I > 35) or keypressed;

end;

( Draw Points Entered Manually )
procedure Draw_Manual_Points;

var
  RA : real;
  Dec : real;
  Magnitude : Integer;

begin
  Pre_Calculator;

  repeat
    Set_Text_Mode;
    Clr_Scr;
    Set_Integer ( 'Magnitude', -2, 10, Magnitude );
    Set_Real ( 'Right Ascension', 0.0, 360.0, RA );
    Set_Real ( 'Declination', -90.0, 90.0, Dec );
    RA := RA * Deg_To_Rad;
    Dec := Dec * Deg_To_Rad;

    ( Change to graphics mode )
    Set_Graphics_Mode;

    Project_Point ( RA, Dec );
    Display ( RA, Dec, Magnitude, false );

  until keypressed;

  until Magnitude < -1.999;

end;

( Draw Test Pattern )
procedure Draw_Test_Pattern;

var
  I : Integer;
  J : Integer;
  Magnitude : Integer;

begin
  Pre_Calculator;

  I := 0;
  repeat
    J := 9;
    repeat
      RA := Int ( I ) * 10. * Deg_To_Rad;
      Dec := Int ( J ) * 10. * Deg_To_Rad;

      Project_Point ( RA, Dec );
      Display ( RA, Dec, Magnitude, false );

      J := J - 1;
    until ( J < -9) or keypressed;

    I := I + 1;
  until ( I > 35) or keypressed;

end;

```

```

    J := 1;
  repeat
    Magnitude := I;
    Display ( Magnitude, I, J, false );
    J := J + 1;
  until (J > 200) or keypressed;
  I := I + 1;
until (I > 640) or keypressed;
end;

```

```

( Generate Graphics Displays )
procedure Generate_Graphics (
  Time_Mode : integer;
  Test_Mode : integer );

```

```

var
  Character : char;

begin
  if Time_Mode = 1 then
    Date_Time ( Time_Mode );

  ( Switch to Graphics Mode )
  Set_Graphics_Mode;

  case Test_Mode of
    1 : Draw_Stars;
    2 : Draw_Grid;
    3 : Draw_Manual_Points;
  end;

  Set_Text_Mode;
end;

```

```

-----
Name:      Main_Menu
Location:  MENU.PAS
Purpose:   Main Menu for telescope simulator
Revision History:
Copyright (c) 1987 by
Doug George, 66 Shetland Way,
Kanata, Ontario, K2M 1S4
-----

```

```

( Master Input Menu )
procedure Main_Menu;

const
    Altazimuth_Mode = true;
    Equatorial_Mode = false;

var
    Operation      : integer;
    RA              : real;
    Dec             : real;
    Scale_Factor   : real;
    Test_Mode       : integer;
    Time_Mode       : integer;
    Menu            : Menu_Type;
    User_Input      : integer;
    Position        : Time_Type;
    First_Time      : boolean;

begin
    First_Time := true;
    Operation := 1;
    Time_Mode := 1;
    Test_Mode := 1;
    Date_Time ( Time_Mode );

    ( Set to initially look directly overhead )
    RA := Siderial;
    Dec := Latitude;
    Scale_Factor := 80.0 * Deg_To_Rad;
    Set_Up ( RA, Dec, Equatorial_Mode, Scale_Factor );

    repeat
        Menu.Max := 5;
        Menu.Title := 'Function';
        Menu.Option: 1 := 'Telescope Control';
        Menu.Option: 2 := 'Horizon Display';
        Menu.Option: 3 := 'System Time Control';
        Menu.Option: 4 := 'Test Mode';
        Menu.Option: 5 := 'Exit';
        if First_Time then
            First_Time := false;
        else
            Get_Menu ( Menu, Operation );

        case Operation of
            1 : ( Telescope Control
                begin
                    ( Set up date and time if not fixed )
                    if Time_Mode = 1 then
                        Date_Time ( Time_Mode );

```

```

    ( Set up and generate graphics )
    Generate_Graphics ( Time_Mode, Test_Mode );
end;

2 : ( Horizon View )
begin
    ( Get necessary data )
    Get_Real ( 'Azimuth', 0, 360, RA );
    RA := RA * Deg_To_Rad;
    Dec := 45 * Deg_To_Rad;
    Scale_Factor := 70 * Deg_To_Rad;
    Set_Up ( RA, Dec, Altazimuth_Mode, Scale_Factor );

    ( Set up and generate graphics )
    Generate_Graphics ( Time_Mode, Test_Mode );

    ( Restore to directly overhead for main display mode )
    RA := Siderial;
    Dec := Latitude;
    Scale_Factor := 80.0 * Deg_To_Rad;
    Set_Up ( RA, Dec, Equatorial_Mode, Scale_Factor );
end;

3 : ( System Time Control )
begin
    Menu.Max := 4;
    Menu.Title := 'Time';
    Menu.Option[ 1 ] := 'Machine Clock';
    Menu.Option[ 2 ] := 'Manual';
    Menu.Option[ 3 ] := 'Siderial';
    Menu.Option[ 4 ] := 'No Change';
    Get_Menu ( Menu, User_Input );

    Time_Mode := User_Input;

    ClrScr;
    repeat
        Date_Time ( Time_Mode );
        GotoXY ( 7, 5 );
        write ( 'Date          = ', Date_String );
        GotoXY ( 7, 7 );
        write ( 'Local Time    = ', Time_String );
        GotoXY ( 7, 9 );
        write ( 'Siderial Time = ', Siderial_String );
        GotoXY ( 1, 24 );
        Delay ( 250 );
    until keypressed or (Time_Mode > 1);
    repeat
    until keypressed;

    ( Restore to directly overhead for main display mode )
    RA := Siderial;
    Dec := Latitude;
    Scale_Factor := 80.0 * Deg_To_Rad;
    Set_Up ( RA, Dec, Equatorial_Mode, Scale_Factor );
end;

4 : ( New Test_Mode )
begin
    Menu.Max := 4;
    Menu.Title := 'Set Test Mode';
    Menu.Option[ 1 ] := 'Stars';
    Menu.Option[ 2 ] := 'Grid';
    Menu.Option[ 3 ] := 'Manual';
    Menu.Option[ 4 ] := 'No Change';
    Get_Menu ( Menu, User_Input );
    if User_Input < 4 then
        Test_Mode := User_Input;
    end;
end;

5 : ;
end;

```



```
initial operation = 5;  
end;
```

Motor Controller Program

```

opt nol
*****
*
* Stepper Motor Control Program
*
* Copyright (c) 1987 by
* Doug George
* 66 Shetland Way, Kanata, Ontario
* 613-591-3119
*
*****

                org $c000                * Start of RAM

port            equ    $6000    * parallel port
timer equ      $a000    * timer port
acia            equ    $8000    * serial port
ram             equ    $c000    * ram area
int             equ    $c7f6    * IRQ vector
fint           equ    $c7f3    * FIQR vector
stack equ      $c7f0    * top of stack

* Global Constants
SidTime equ     $cfc3    * 4x siderial rate - 1
BaudTime equ    $0034    * 1/2 of 16 * 600 baud clock - 1
MotTime equ     $01c0    * motor interrupt (tuned for max speed)
BackAdj equ     $d0      * Adjustment to Dec to remove backlash
Normal equ      0        * State - Normal tracking mode
Slewing equ     1        * State - Slewing
SlewReq equ     -1       * State - Slew Requested

start
    bra main

* Stepper Tables

HalfStep        fcb $f9,$f9,$f8,$f8,$fc,$fc,$f4,$f4
                fcb $f6,$f6,$f2,$f2,$f3,$f3,$f1,$f1
FullStep        fcb $f9,$f9,$f9,$f9,$fc,$fc,$fc,$fc
                fcb $f6,$f6,$f6,$f6,$f3,$f3,$f3,$f3
OverStep        fcb $09,$f0,$f0,$f0,$0c,$f0,$f0,$f0
                fcb $06,$f0,$f0,$f0,$03,$f0,$f0,$f0

* Miscellaneous Global/Permanent Variables
* NOTE: storage order critical on RA and Dec variables

State fcb 0      * Present state
RAPnt fcb 0      * Present RA Motor position
DecPnt fcb 0     * Present Dec Motor Position
RADist fdb 0     * Distance Travelled Indicator
                fcb 0
RACoarse fdb 0   * Dec coarse position control.
                fcb 0
RAFine fdb 0     * RA fine position control.
                fcb 0
RABack fcb 0     * RA Backlash Flag
SubRA fcb 0      * RA Subcounter
DecDist fdb 0    * Distance Travelled Indicator
                fcb 0
DecCoarse fdb 0  * Dec coarse position control.
                fcb 0
DecFine fdb 0    * Dec fine position control.
                fcb 0
DecBack fcb 0    * Dec Backlash flag
SubDec fcb 0     * Dec Subcounter
RATime fcb 0     * RA timer
DecTime fcb 0    * Dec timer
SubMask fcb 0    * Subcounter mask

```

```

*****
*
* Motor Step and I/O Routines
*
*****

```

```
* procedure Step
```

```
UpStep equ 1
```

```
OwnStep equ -1
```

```
* Input Parameters
```

```
Dir fcb 0 * 0 means RA, 1 means Dec
```

```
UpDown fcb 0 * Step Up or Down
```

```
StepSize fcb 0 * Half or Full Step
```

```
fcb 0
```

```
* Motor State Variables
```

```
motRA fcb 0 * present RA motor state
```

```
motDec fcb 0 * present Dec motor state
```

```
step
```

```
pushs a,x
```

```
test Dir * Check the direction
```

```
bne sDec
```

```
RA
```

```
ldx #RAPnt * Point X to RAPnt
```

```
bra sDirDne
```

```
Dec
```

```
ldx #DecPnt * Point X to DecPnt
```

```
sDirDne
```

```
lda ,x
```

```
adda UpDown * Add UpDown to position
```

```
anda #SOf * Reset if at end of table
```

```
sta ,x
```

```
ldx StepSize
```

```
* Save
```

```
* Point to stepper table
```

```
lda a,x
```

```
* Load value to output
```

```
test Dir
```

```
* Check direction again
```

```
bne sSDec
```

```
sRA
```

```
cmpa #$10 * RA motor output
```

```
bne sSRANorm * Check for clearing overpulse
```

```
ora motRA
```

```
sSRANorm
```

```
sta port+2 * Store to RA port
```

```
sta motRA
```

```
bra sNone
```

```
sSDec
```

```
cmpa #$10 * Dec motor output
```

```
bne sSDecNorm * Check for clearing overpulse
```

```
ora motDec
```

```
sSDecNorm
```

```
sta port * Store to Dec port
```

```
sta motDec
```

```
step
```

```
pulls a,x
```

```
rti
```

```
* procedure Read
```

```
* parameter passed in register A
```

```
read
```

```
lda acla
```

```
lsta
```

```
bcc read
```

```
lda acla+1
```

```
rti
```

```
* procedure readWait
```

```
* parameter passed in register A
readD
```

```
    bsr read
    cmpa #'9
    bhi rda1pha
    suba #'0
    bra rdout
rda1pha
    suba #'A-10
rdout
    rts
```

```
* procedure readByte
* parameter passed in register A
readB
```

```
    bsr readD
    asla
    asla
    asla
    asla
    pshs a
    bsr readD
    ora ,s
    leas 1,s
    rts
```

```
* procedure ReadWord
* parameter passed in register D
readW
```

```
    bsr readB
    tfr a,b
    bsr readB
    exg a,b
    rts
```

```
*****
*
* Interrupt Service Routines
*
*****
```

```
Speed1 equ    8      * Time delay between steps
Speed2 equ    5
Speed3 equ    3
Speed4 equ    2
Speed5 equ    1
```

```
Dist1 equ    1      * Distance ranges for different speeds
Dist2 equ    2
Dist3 equ    8
Dist4 equ   13
Dist5 equ   20
```

```
* Interrupt Service Routines
ISR
```

```
    pshs x,y,u,d,cc
    lda timer+1      * Read timer interrupt flags
    bita #$02
    beq iNoSid       * Branch if not siderial int
    lda timer+4      * Clear siderial timer (#2)
    bsr sidIsr       * Execute siderial time isr
    bra iIntDon      * Finish up
```

```
iNoSid
    bita #$01
    beq iNoMot       * Branch if not Motor interrupt
    lda timer+2      * Clear motor timer (#1)
    bsr motIsr       * Execute motor isr
    bra iIntDon      * Finish up
```

```
iNoMot
    lda #$ff         * illegal interrupt
```

```

        sta port
        sta port+2
        bra iNoSid

;IntDor
        puls x,y,u,d,cc
        rti

* Siderial Time ISR
sidCnt fcb 0      * divide by four counter

sidIsr
        lda sidCnt      * Load siderial counter
        inca
        anda #503      * Increment modulo-4
        sta sidCnt
        one noCnt
        ldx RAFine      * Increment RA if timeout
        leax 1,x
        stx RAFine

noCnt
        rts

* RA/Dec Motor Timing ISR
RADec fcb 0      * 0 means RA cycle, 1 = Dec

motIsr
        lda RAdec      * Determine whether RA or Dec
        inca
        anda #501
        sta RAdec
        sta Dir      * Store direction for motor
        bne DecCyc

RAcyc
        ldy #RATime      * Y points to RA Timer
        ldx #RAFine      * X points to RA Fine Distance
        bra Cycle

DecCyc
        ldy #DecTime      * Y points to Dec Timer
        ldx #DecFine      * X points to Dec Fine Distance

Cycle
        dec y      * Decrement timer
        lbne CycEnd      * Branch out if no timeout

        tst State      * Check present state
        opl NoReq      * Check for new slew request

NoReq
        lda motRA      * Check for odd step position
        ora motDec      * on RA or Dec
        anda #51      * LSB indicates even or odd
        lbne Norm      * Continue normal stepping if odd

Req
        lda #Slewing      * Indicate state is slewing
        sta State
        clr SubDec      * Set up sub-counters for divide-by-32
        clr SubRA
        lda #51f
        sta SubMask
        lda #50      * Clear distance travelled
        std RADist
        std DecDist

NoReq
        tst State      * Determine if slewing
        ple Norm      * Go to normal stepping
        ldd RACoarse      * Check for zero RA
        one MoreSlew      * Continue slewing
        ldd DecCoarse      * Check for zero Dec
        one MoreSlew      * Continue slewing

ActNorm
        clr State      * Return to normal state
        ldd #HalfStep      * Set back to half steps
        std StepSize

```

```

        lda #S1
        sta SubMask      * Set subcounter mask to divide-by-two
        bra Norm          * Go to normal stepping

MoreSlew
        ldd -3,x          * Load distance remaining
        bpl RemPos        * Take absolute value
        coma
        comb
        addd #S1

RemPos
        cmpd -6,x          * Compare with distance travelled
        bmi RemSmall      * Take smaller of two numbers
        ldd -6,x          * Distance is smaller

RemSmall
        cmpd #Dist1       * If smaller than Dist1, take Speed1
        blt S1
        cmpd #Dist2       * If smaller than Dist2, take Speed2
        blt S2
        cmpd #Dist3       * If smaller than Dist3, take Speed3
        blt S3
        cmpd #Dist4       * If smaller than Dist4, take Speed4
        blt S4

S5
        ldd #OverStep      * Set to overvoltage stepping
        std StepSize
        lda #Speed5        * Take Speed5
        bra SetSpeed

S4
        ldd #OverStep      * Set to overvoltage stepping
        std StepSize
        lda #Speed4        * Take Speed4
        bra SetSpeed

S3
        ldd #OverStep      * Set to overvoltage stepping
        std StepSize
        lda #Speed3        * Take Speed3
        bra SetSpeed

S2
        ldd #FullStep      * Set to full stepping
        std StepSize
        lda #Speed2        * Take Speed2
        bra SetSpeed

S1
        ldd #FullStep      * Set to full stepping
        std StepSize
        lda #Speed1        * Take Speed1

SetSpeed
        sta ,y            * Store in timer
        ldb #-3           * Indicate access Coarse counter
        bra Go

Norm
        clrb              * Indicate access Fine counter
        lda #Speed1        * Set speed to 1
        sta ,y            * Store in timer

Go
        ldy b,x            * Y gets Distance (b = Fine or Coarse)
        beq mBakLsh        * Branch out if nowhere to go
        bmi mDown          * If negative, go down

mUp
        lda #UpStep        * Set for step up
        sta UpDown
        lbrs step          * Perform the step

        lda 4,x            * Check subcounter
        inca              * Increment
        anda SubMask       * Wrap-around mask
        sta 4,x            * Store back in subcounter
        bne CycEnd         * If not at zero, don't adjust position
        leay -1,y          * Decrement Distance To Go
        sty b,x            * Save in Coarse or Fine Distance

        ldd -6,x          * Set distance travelled (PA/100 x 10)
        addd #S1          * Increment it and save

```

```

        std -6,x
        bra CycEnd

*Down
        lda #DwnStep      * Set for step down
        sta UpDown
        lbrs step          * Perform the step

        lda 4,x            * Check subcounter
        inca               * Increment
        anda SubMask       * Wrap-around mask
        sta 4,x            * Store back in subcounter
        bne CycEnd         * If not at zero, don't adjust position
        leay 1,y            * Increment Distance To Go
        sty b,x            * Save in Coarse or Fine Distance
        pshs cc             * Save condition of distance to go = 0
        ldd -6,x            * Get distance travelled (RA/Dec Dist)
        addd #51            * Increment it and save
        std -6,x
        puls cc
        bne CycEnd        * Don't perform backlash adj if not zero

*Backlash
        tst State           * Don't perform backlash adj if slewing
        bne CycEnd
        tst 3,x             * Check for backlash adjust
        beq CycEnd
        leay BackAdj,y      * Reverse the motor
        sty ,x
        clr 3,x             * Clear the flag

*End
        rts                * Exit

*****
*
* Mainline
*
*****

Main
        lds #stack          * Stack near end of RAM

        lda #57e            * Set up FIRO vector
        sta fint            * using a branch instruction
        add #1sr            * in RAM
        std fint+1

        clr port+1          * Initialize 6821 parallel port
        clr port+3
        lda #5ff
        ldb #52e
        sta port
        stb port+1
        sta port
        stb port+2
        stb port+3
        sta port+2

        ldd #BaudTim        * Set up baud rate generator
        std timer+6         * on timer #3
        ldd #SidTime        * Set up 4x sidereal rate
        std timer+4         * on timer #2
        ldd #MotTime        * Set up 1ms motor rate
        std timer+2         * on timer #1

        clr timer+1         * Select CR3
        lda #582            * Set up timer #3
        sta timer
        lda #543            * Select CR1; set up timer #2
        sta timer+1
        lda #542            * Timers on; set up timer #1
        sta timer

        lda #503            * Reset ACIA
        sta acia
        lda #511            * 8-bit, 2-stop, /16 clock

```

```

sta acia
lda acia+1      * Clear port of any data

clr RAPnt      * Clear motor positions
clr DecPnt
lda #S1
sta RAtime     * Initialize RA and Dec timers
sta DecTime
ldx #S0        * Clear coarse and position counters
stx RACoarse
stx DecCoarse
ldx #-BackAdj  * Set up RAFine, DecFine to remove
stx RAFine     * backlash upon power-up
stx DecFine
lda #S1
sta RABack     * Set backlash adjust flags
sta DecBack
ldx #HalfStep  * Initialize step size
stx StepSize
lda #S1
sta SubMask    * Set to adjust counter on every second step
lda #Normal    * Set up normal tracking mode
sta State

mloop
andcc #Sbf     * Enable clock interrupts
noLoop
lbr read       * get char in reg a

cmpa #'D       * compare to a 'D'
bne noD        * branch to next test if not
lbr readW      * get digit in reg d
tst DecBack    * check for already adjusted
bne noDBack
add #BackAdj
inc DecBack    * Indicate adjustment
noDBack
orcc #S40      * temporarily disable int
add DecFine    * Add it to Dec Fine position
std DecFine
bra mloop      * back to start
noD
cmpa #'R       * Same for Right Ascension
bne noR
lbr readW
tst RABack     * check for already adjusted
bne noRBack
add #-BackAdj
inc RABack     * Indicate adjustment
noRBack
add RAFine
std RAFine
bra mloop
noR
cmpa #'M       * Check for High-Speed Slew
bne noM

lbr readW      * Get Coarse RA adjustment
std RACoarse   * Put in counter

lbr readD      * Get Fine RA adjustment
ldx RAFine
leax a,x       * Add LSD and backlash adjust to RAFine
leax #-BackAdj,x

lbr readW      * Get Coarse Dec adjustment
std DecCoarse  * Put in counter

lbr readD      * Get Fine Dec adjustment
ldy DecFine
leay a,y       * Add LSD and backlash adjust to DecFine
leay #-BackAdj,y

orcc #S40      * Disable interrupts

```



```

lda #SlewReq      * Indicate Slew Request
sta State
stx RAFine        * Store RAFine
sty DecFine       * Store DecFine
inc RABack        * Indicate RA backlash adjustment
inc DecBack       * Indicate Dec backlash adjustment
bra mloop

;M
cmpa #15          * Stop motion
bne noS
orcc #540
ldd #50
std DecFine
std RAFine
std DecCoarse
std RACoarse
clr RABack
clr DecBack
bra mloop

;M
bra nola

end

```

END

1|7·0|2·90

FIN