ESV Workstation

Spaceball User's Manual

EVANS & SUTHERLAND COMPUTER CORPORATION
Salt Lake City, Utah

DOCUMENTATION WARRANTY:

PURPOSE: This documentation is provided to assist an Evans & Sutherland trained BUYER in using a product purchased from Evans & Sutherland. It may contain errors or omissions that only a trained individual may recognize. Changes may have occurred to the hardware/software, to which this documentation refers, which are not included in this documentation or may be on a separate errata sheet. Use of this documentation in such changed hardware/software could result in damage to hardware/software. User assumes full responsibility of all such results of the use of this data.

WARRANTY: This document is provided, and Buyer accepts such documentation, "AS-IS" and with "ALL FAULTS, ERRORS, AND OMISSIONS." BUYER HEREBY WAIVES ALL IMPLIED AND OTHER WARRANTIES, GUARANTIES, CONDITIONS OR LIABILITIES, EXPRESSED OR IMPLIED ARISING BY LAW OR OTHERWISE, INCLUDING, WITHOUT LIMITATIONS, ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. BUYER FURTHER HOLDS SELLER HARMLESS OF ANY DIRECT OR INDIRECT DAMAGES, INCLUDING CONSEQUENTIAL DAMAGES.

Part Number: 517941-201 AA
April, 1991

Table of Contents

1. Introduction

This document complements the *Spaceball Model 1003 Product Reference Manual*, Document Number D1003, Spatial Systems Pty Limited, April 1988. The *Product Reference Manual* should be read first as a tutorial introduction to the Spaceball Option. This guide is arranged as follows:

- "Chapter 1" (this chapter) contains installation instructions for Spaceball.
- "Chapter 2" describes X Spaceball implementation.
- "Chapter 3" describes ES/PSX Spaceball implementation.
- "Appendix A" contains an ES/PSX example program.

Overview

Spaceball is an interactive device consisting of a stationary ball mounted on a base. It senses applied force and torque. Eight programmable buttons are located on the upper face, which can be used as function keys, and another programmable button is located on the front of the ball, which can be used as a pick button.

Spaceball has six degrees of freedom. By pushing and twisting the stationary ball, Spaceball simultaneously senses the forces along, and the torques around, the *x*,*y*,*z* axes of its coordinate system. These six parameters are processed by Spaceball and output as numeric vectors and matrices. They can be used to control any function you specify.

Installation

1) Plug the 9 V power adapter into the backshell of the RS-232-C connector.
2) Plug this end of the cable into the filter, and then into an unused port (**tty0** or **tty3**) of the ESV Workstation.
3) Plug the other end of the cable (9-pin connector) into Spaceball.
4) Apply power to Spaceball. Spaceball should generate two short beeps. If Spaceball does not beep, check the cable connections.
5) If Spaceball beeps an SOS signal, the unit is malfunctioning.

Software Considerations

- The **tty** channel used must be set to be readable and writable. Login as **root** and enter the following command:

  chmod /dev/ttyn

  where **n** is the appropriate number (**0** or **3**).

- The third field in the **/etc/inittab** file for **t0** or **t3** must be set to **off**.

- The default internal speed setting for Spaceball is 9600 baud. The default speed for the **tty** ports set by the UNIX kernel is also 9600 baud. If the internal speed of Spaceball has been changed, the port speed must also be changed. This can be done with the **stty** command as follows:

  /bin/stty 4800 </dev/tty0 (System V)

  /bsd43/bin/stty 4800 >/dev/tty0 (BSD 4.3)

2. X and PHIGS Implementation

This chapter describes the operation of Spaceball when it is used in the X environment and when it is used with PHIGS input.

X Input Extension

The X Input Extension is used to send data from Spaceball to the ESV Workstation. Refer to chapter 2 in the *ESV Workstation Reference Manual [2.0]* for more information about using the X Input Extension with Spaceball.

PHIGS Implementation

PHIGS input can be used to access Spaceball. For information on how to do this, see the "PHIGS Input with ESV Devices" section in chapter 1 of the *ESV Workstation Reference Manual [2.0]*.

Utility Routines

The utility routines are used to process events from Spaceball whether it is used in an X environment or with PHIGS input. Spaceball is a sophisticated device, and it can be set to perform data processing before generating any events. However, Spaceball cannot be in a different state in each different window. Spaceball sends raw data, and the utility routines are used to provide some of the Spaceball processing features in a non-window environment. These routines are located in the **/usr/lib/libXEandSext.a** library.

SbUnpackMatrix

SbUnpackMatrix(*data, matrix*)
Int ***data[6]*;**
SBmatrix3 ***matrix*;**

This routine expects a Spaceball motion data array. This array can be the axis data array from an XDeviceMotion Event structure or the decoded axis information from a PHIGS **pget_string** call. It returns, in the space pointed to by ***matrix***, a 4x4 homogenous rotation matrix. You must be careful to initialize ***matrix*** to the identity matrix since not all matrix elements are touched.

The rotation matrix is formed using the compact rotation information in the data array and the mode information set by the other utility routines. The **SBmatrix3** structure is identical to a PHIGS **Pmatrix3** structure.

SbRotAxisEnable

SbRotAxisEnable(*x_enable, y_enable, z_enable*)
Int ***x_enable;***
Int ***y_enable;***
Int ***z_enable;***

This routine enables/disables Spaceball rotation data for each axis. If the value for a flag is non-zero then that corresponding axis will affect the matrix returned by **SbUnpackMatrix**. Likewise if the flag is zero then that axis will not contribute to the rotation matrix.

Note: Spaceball is a very sensitive device, and it is very difficult to twist it without somewhat affecting all rotation parameters.

SbRotSensitivity

SbRotSensitivity(*factor*)
float ***factor;***

factor can be used to scale the incoming rotation data. This allows you to change the amount of rotation according to the amount of twist provided.

A ***factor*** value between 0.0 and 1.0 scales down the rotation. A ***factor*** value of 0.0 has the effect of disabling all rotations. Values greater than 1.0 scale up the rotation. A negative ***factor*** value reverses the direction of rotation.

SbRotDominantMode

SbRotDominantMode(*flag*)
Int ***flag;***

This routine enables/disables *dominant* mode. In *dominant* mode Spaceball uses the raw rotation data to find the axis that is being twisted the most (the dominant axis). When the event is unpacked using **SbUnpackMatrix**, the dominant axis is the only active axis. Of course **SbUnpackMatrix** will still pay attention to the axis enable and sensitivity settings.

This might be used in an application requiring only *y*-axis rotations. Set the axis enable using **SbRotAxisEnable(false, true, false)**. However, when you twist Spaceball on another axis it will still affect the *y*-axis somewhat. By setting **SbRotDominantMode(true)**, when you twist another axis, it is enabled by dominant mode but disabled with by the current axis enable mode. This results in an identity matrix being returned by **SbUnpackMatrix**. So you only get the *y*-axis rotation when you intend to twist around the *y*-axis.

SbDominant

SbDominant(*vector*)
SBvector3 *vector*;

SbDominant takes a structure of three floating point numbers and zeros out the two smaller elements. The comparison for the largest element (the dominant one) is done using the absolute magnitude of the entries.

This routine is used by **SbUnpackMatrix** to select the dominant rotation element when dominant mode is enabled.

SbInitialize

SbInitialize()

This routine initializes internal variables and sets up the processing mode for **SbUnpackMatrix**. The initial processing mode is all axis enabled, sensitivity factors set to 1.0, and dominant mode disabled.

SbInitialize can be called at any time, but to ensure consistent behavior, it should be called in an application immediately after attaching Spaceball.

3. ES/PSX Implementation

Overview

Spaceball is implemented as a **tty** device connected to the back panel of the ESV Workstation. When connected in this manner, Spaceball is unavailable for use as an X device and can only be used by one ES/PSX process at a time. To inform ES/PSX of the port that is used by Spaceball, you should set the environment variable ***PSX_SPACEBALL***. For example,

```
setenv PSX_SPACEBALL /dev/tty3
```

The vectors and matrices output from Spaceball are input to an ES/PSX function network, which uses the data to control the functions you select. The programmable buttons allow you to enable or disable any of the input data, change the function of any of the input data, or pick any data.

Since Spaceball senses the three forces and three torques simultaneously, it provides unlimited control over the positioning and orienting of an object in space on ES/PSX. Spaceball output can also be used to control other functions such as object scale and color. By connection to the appropriate function, Spaceball can emulate other interactive devices, such as a mouse, control dials unit, function buttons unit, and data tablet.

Principles of Operation

Communication

Spaceball communicates with ES/PSX over a **tty** interface by sending messages to, or receiving commands from ES/PSX.

Messages and commands consist of **Qpacket**s ending with a carriage return. The first character of each **Qpacket** identifies the type of message or command. The format of the data for each message or command type is described in this chapter.

Data within each **Qpacket** can be in one of several protocols determined by the communication mode set in Spaceball. ES/PSX always sets the communication mode to be the *binary* mode. It removes any escape sequences in data received from Spaceball and inserts escape sequences in data sent to Spaceball.

Ball Functionality

Spaceball senses force and torque, converts the information into a format that you specify, and sends the data in a **Qpacket**. The presence and format of the data is determined by the *data* mode. It is possible to specify that *translation* and/or *rotation* data be sent from Spaceball. No data will be sent unless *translation* and/or *rotation* data is enabled.

Translation Data

If *translation* data is enabled, the relative force applied in each of the *x,y,z* directions is sent, along with a multiplier related to the length of time the force had been applied since the last data message. This information is converted to a 3D vector by the function processing Spaceball data.

Rotation Data

If *rotation* data is enabled, the relative torque around each axis is sampled and the data is sent in a format which you specify.

Vector Rotation Format

When this format is specified, the torque around each axis is sent, along with a multiplier related to the period since the last data message. This format is similar to the format of *translation* data and is converted by ES/PSX into a 3D vector. The vector is the change in rotation torque since the last data message. Since the *rotation* data format and the *translation* data format are identical, they can be used interchangeably for similar purposes.

Delta Rotation Matrix Format

This data format causes a 3x3 rotation matrix to be sent, which is the rotation change which would have been applied to an object by the torque sensed since the last message sent.

Absolute Rotation Matrix

This data format causes a 3x3 rotation matrix to be sent, which is the accumulated rotation effect which would have been applied to an object by the torque sensed. A command can be sent to Spaceball to initialize the internal *absolute rotation matrix*. Torque applied to Spaceball then causes updated accumulated rotation matrices to be sent. This implies that these 3x3 matrices can update a *rotation* node directly without further computation in ES/PSX.

Constraining the Data

The *data* mode of Spaceball can be set such that only linear or planar freedom is allowed. A *translation* or *rotation* freedom vector can then be sent which defines the linear axis, or the normal to the plane for the freedom allowed.

Scaling the Data

The 3D vectors received as a result of *translation* data or the vector form of *rotation* data, can be scaled easily by multiplying each component of the vector by a factor which provides the performance desired.

Matrix forms of *rotation* data are more difficult to manage. Spaceball can be instructed to handle this problem itself by modifying the sensitivity of rotation matrices. The *spin rate* factor can be adjusted to achieve the required performance. The *spin rate* cannot be used to scale translation or rotation vectors.

Coordinate Systems

The coordinate system of Spaceball should match that of the object. Spaceball can be instructed that the coordinate system of the object is either left-handed or right-handed, which defines the positive *z*-direction. In addition, the orientation of Spaceball can be updated in a more general sense by sending a 3x3 *orientation* matrix. This can cause the orientation of Spaceball to correspond to a top view or side view.

It is also possible to specify whether the coordinate systems for Spaceball, the freedom vectors, and Spaceball output are in object or world coordinate systems, called local or parent, respectively. This information is sent in the same command which sets the freedom vectors.

Zeroing

It is possible to reset the initial state of Spaceball. When this happens, all further data will be relative to the torque and force applied when Spaceball was zeroed. This can allow simple animation by zeroing Spaceball after applying some torque. When the ball is released, data will be continuously sent in reverse of the torque applied.

Sensitivity

Spaceball can be made more sensitive to the touch by updating the *null radius* and *feel*. If the *null radius* value is made smaller, less effort will be required to cause Spaceball to output data. This can have a negative effect, since unnecessary data can be sent if the value is too small. Adjusting the *feel* causes linear to cubic response when increased torque and force is applied.

Keypad Functionality

When a keypad button is pressed or released, a message is sent to the ESV Workstation which causes an integer to be generated indicating the number of the button pressed or released. Buttons one through eight generate integers one through eight respectively. The pick button generates an integer nine. Output <2> of the **sbin** function generates integers when a button is pressed,

and output <3> of the **sbin** function generates integers when a button is released.

Beeper Functionality

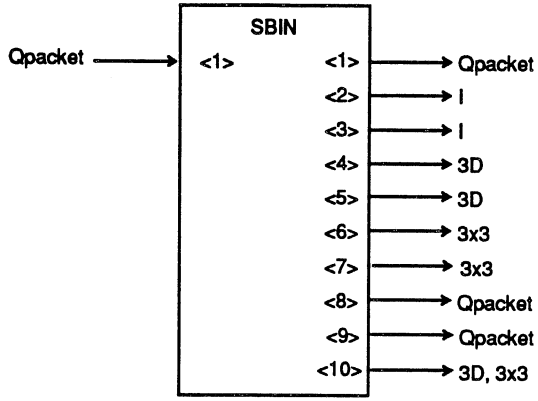
The beeper interface has a 30-note queue. Each note contains a flag, indicating sound or silence, and a five-bit integer duration in 1/32^{nds} of a second.

Spaceball Functions

SBIN

Type:

Intrinsic User Function - Miscellaneous



Purpose:

This function converts data from Spaceball into ES/PSX format and synchronizes communication with Spaceball.

Description:

Input

<1> — connected from Spaceball and also from the **sbout** function

Outputs

<1> — connected to Spaceball
<2> — integer keypad button pressed
<3> — integer keypad button released
<4> — delta translation vector
<5> — delta rotation vector
<6> — delta rotation matrix
<7> — absolute rotation matrix
<8> — error/warning messages
<9> — other packets from Spaceball
<10> — decoded messages from Spaceball

Notes:

1) Input <1> acts as the function trigger. It accepts **Qpackets** containing commands and queries for Spaceball and messages from Spaceball. If it receives a Boolean true, data messages are sent through outputs <9> and <10>.

   Input <1> is connected to output <1> of **B4$**, the input function for port 4. It is also connected to output <1> of the **sbout** function, so that it can coordinate synchronization of commands to Spaceball with messages from Spaceball. Never send any commands directly to this input. Commands for Spaceball must first be processed by the **sbout** function to identify the data as coming from ES/PSX rather than from Spaceball.

2) Output <1> is connected to input <1> of **O4$**, the output function for port 4.

3) An integer corresponding to the button pressed or released is sent through output <2> or output <3>, respectively. The pick button is button nine.

4) The presence of data on outputs <4> through <7> depends on the **mode** setting:

   - If **translation** data is enabled, it is sent through output <4>.
   - If **rotation** data is enabled and is to be in delta vector format, it is sent through output <5>.
   - If **rotation** data is enabled and is to be in delta matrix format, it is sent through output <6>.
   - If **rotation** data is enabled and is to be in absolute matrix format, it is sent through output <7>. Messages returned after updating or querying the absolute matrix of Spaceball are also sent through output <7>.

5) Error messages from Spaceball are sent through output <8>. Errors discovered by the function during processing of messages from Spaceball or commands to Spaceball, are also sent through output <8>.

6) The following messages from Spaceball are sent through output <9>:

   % (echo)
   A (absolute rotation matrix)
   B (beeper)
   D (data - sent only if enabled)
   F (feel)
   H (help)
   M (mode)
   N (null radius)

O (orientation)
R (rotation mode)
S (spin rate)
T (translation mode)

7) Absolute and orientation matrices, and translation and rotation freedom vectors, are decoded and sent from output <10>. These messages are also copied unchanged and sent from output <9>. Output <10> is a **Qpacket** containing the input code type followed by decoded matrices and vectors. The following table lists the code sent in the **Qpacket** and the associated data type which will immediately follow.

Table 1. Output <10> message types

Message Type	Qpacket Code	Data Type
Absolute matrix	'A'	3x3
Orientation matrix	'O'	3x3
Translation freedom vector	'T'	3D
Rotation freedom vector	'R'	3D
If enabled:		
Translation data	'DT'	3D
Vector rotation data	'DR'	3D
Delta rotation data	'DR'	3x3
Absolute rotation data	'DR'	3x3

To use this output, connect it first to a print function and then to the host_message function.

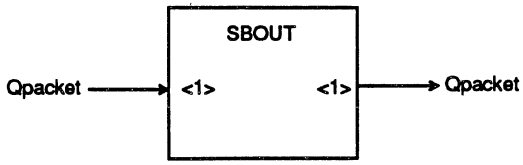
8) Since the **sbin** function has no way of knowing when an application starts and finishes, the application must initialize Spaceball to the state desired. When the application terminates, it should always disable transmission of data by changing the data mode.

A program should not assume that the **sbin** function will be in any known state. When it starts, it will typically set the data mode, and, if using absolute rotation data, send an identity absolute matrix. The **null radius, feel, spin rate**, and **relative translation sensitivities** can also be set.

SBOUT

Type:

Initial Function Instance - Data Output



Purpose:

This function identifies **Qpackets** to be sent to Spaceball as coming from ES/PSX. Since the **sbin** function coordinates communication with Spaceball, it processes both packets coming from Spaceball and going to Spaceball. It must be able to tell which type of packet it is processing.

Description:

Input

<1> — commands for Spaceball

Output

<1> — connected to **sbin** function

Notes:

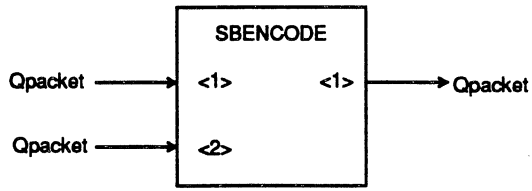
1) Input <1> can receive commands directly from function networks or can be connected from the **sbencode** function.

2) Output <1> is connected to input <1> of the **sbin** function.

SBENCODE

Type:

Initial Function Instance - Data Output



Purpose:

This function encodes ES/PSX format data into Spaceball format, and concatenates the encoded data to the **Qpacket** on input <1>. It outputs encoded commands to be sent to Spaceball.

Description:

Inputs

<1> — command for Spaceball to be concatenated with encoded data from <2>

<2> — data to be encoded from ES/PSX format into Spaceball format

Output

<1> — connected to **sbout** function

Notes:

1) Input <1> is a **Qpacket** which is to be concatenated with the encoded data from input <2>.

2) Input <2> is ES/PSX data which is to be encoded into Spaceball format. Valid data types depend on the command code on input <1>. The following table lists the codes accepted on input <1> and the corresponding data type expected on input <2>.

Table 2. Codes accepted on input <1>

Input <1>	Input <2>	Command Type	Processing
'A'	3x3	Absolute matrix	Converted to a compact rotation matrix
'O'	3x3	Orientation matrix	Converted to a compact rotation matrix
'F'	I or R	Feel	Converted to printable ASCII character
'N'	I or R	Null radius	Converted to printable ASCII character
'R'	V3D	Rotation freedom vector	Converted to 16-bit vector
'S'	V2D	Spin rate (exponent,mantissa)	Converted to 2 printable ASCII characters
'T'	V3D	Translation freedom vector	Converted to 16-bit vector
'X'	V3D	Translation sensitivity vector	Converted to 16-bit vector

3) It is good ES/PSX programming practice to put a sync function between the **sbencode** function and each of its inputs. For example, if a single keypad button is to send an absolute matrix and a rotation freedom vector, there should be two sync functions. One sync function will gate the command and matrix for the absolute matrix, and the other sync function will gate the command and vector of the freedom vector. This will prevent the matrix from arriving with the **freedom** command, and vice versa. The sync functions will also prevent the **sbencode** function from getting out of synchronization with the network, especially during debugging of an application.

Spaceball Function Network

Figure 1 shows the function network created in the ES/PSX runtime.

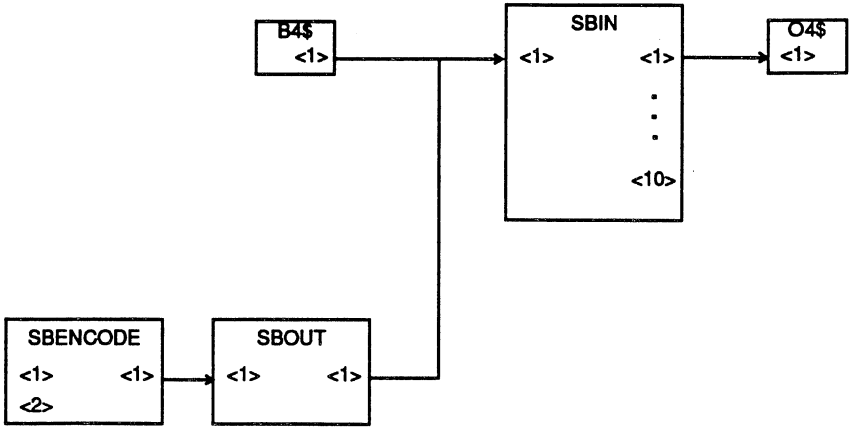


Figure 1. Spaceball function network

Command and Message Codes

Spaceball communicates using single byte command or message codes with appropriate data associated with each code. Messages from Spaceball are **Qpackets** containing escape sequences, and are terminated by a carriage return. Commands and queries sent to Spaceball are **Qpackets** which must be preprocessed by the **sbout** function to identify them as coming from function networks.

Commands and queries to be sent to Spaceball are edited as much as possible, escape sequences are inserted where necessary, and a carriage return is added to the end of the data before sending it to Spaceball as a **Qpacket**. Most commands, and all queries cause a response to be returned from Spaceball. When a message response is to be returned from Spaceball, a flag is set indicating that all further commands and queries are to be queued until the response is received. When the response message is received, the wait-for-response flag is cleared, and the next queued command or query is sent. This synchronization of commands and their responses avoids errors which can occur due to transmitting and receiving at the same time.

Messages from Spaceball are processed by removing any escape sequences, by saving any changes in state so that Spaceball can be initialized to the current state when necessary, and by decoding data and routing it to the appropriate output. If *data* messages are enabled (see **mode** command), Spaceball is polled for the next data message.

- Commands to Spaceball are of the form:

  "Upper Case Letter"<data><CR>

- Queries of Spaceball are of the form:

  "Lower Case Letter"<CR>

  (no data, return the corresponding Upper Case Message in response)

- Messages from Spaceball are of the form:

  "Upper Case Letter"<data><CR>

Most commands and messages always contain data in a printable format. When the data is in this format, the top two bits are ignored, and the lower six bits form an unsigned binary value, called a *printable ASCII character*.

Only the *binary* communication mode is supported by Spaceball functions. Error messages will be issued if attempts are made to change to another mode.

For most commands and all queries, the wait-for-response flag is set and then reset only when the corresponding response message is returned. Cases are explicitly mentioned below where the command does not set a wait-for-response flag, or where the message always resets the wait-for-response flag.

For most messages, the processing involves saving the data so that it can be used later to initialize Spaceball to the current state if Spaceball is reset. The message is then copied and sent from output <9>. If the message contains data in an internal Spaceball format, it is also decoded and sent from output <10>. The message code type is sent in a **Qpacket** from output <10>, followed by the decoded vector or matrix.

Data messages are handled differently. They are decoded and the data sent from the output corresponding to the type and format of the data received. If a Boolean true is sent to input <1> of Spaceball, data messages will also be sent to outputs <9> and <10>.

Following is a summary of the command and message codes.

Space, Tab (null command/message)

Description

These codes indicate a **null** command or message.

Command Processing

The **sbin** function always clears the wait-for-response flag, and sends the data to Spaceball. A space is a good way to initialize the **sbin** function and wake Spaceball up after it has been powered on. Spaceball counts receipt of these messages and resets the count upon receiving **h?**.

Message Processing

The **sbin** function always clears the wait-for-response flag and copies the message to output <8>. The messages are received in response to an **echo** command.

% (echo)

Description

This code indicates a return of the data from Spaceball.

Command Processing

The **sbin** function sends the command to Spaceball, and Spaceball returns the data with a message starting with ' ' (space).

? (Invalid packet)

Description

This code indicates an invalid packet.

Command Processing

The **sbin** function sends the illegal command message to output <8>. Spaceball counts the receipt of these messages and resets the count upon receiving **h?**.

Message Processing

The **sbin** function always clears the wait-for-response flag and copies the message to output <8>.

@RESET (reset)

Description

This code resets Spaceball to the initial power on state.

Command Processing

The **sbIn** function sends the **space** command to wake up Spaceball, and then sends the **reset** command. Spaceball resets itself to the initial values, and then sends the following messages:

```
@1 Spaceball alive and well after a software reset.
@2 Firmware version x.yy created on dd-Mon-19yy.
```

Message Processing

The `@1` message is ignored. The `@2` message causes the **sbIn** function to send any messages it has saved in a private queue, setting Spaceball to the state prior to reset. It also will resend any commands for which a message response has not been received.

Notes

The **sbIn** function attempts to update Spaceball to the last known state whenever Spaceball is reset. For this reason, it is best to initialize Spaceball with the desired values instead of sending the **reset** command.

A (absolute rotation matrix)

Description

This code indicates an absolute rotation matrix.

Query Processing

Spaceball returns the current value of the absolute rotation matrix.

Command Processing

The **sbin** function sends the command, and Spaceball updates the internal absolute rotation matrix and returns the current value of the absolute rotation matrix.

Message Processing

The **sbin** function saves the message in a private queue for reset and copies the message to output <9>. A **Qpacket** with code **'A'** and the decoded matrix is sent from output <10>. It also sends the decoded matrix from output <7> as if it were an absolute data matrix.

Data Format

```
<Compact Rotation Matrix>
```

(Initialized to "$000000000000")

Notes

absolute rotation matrix commands can be created by sending an **'A'** to input <1> of the **sbencode** function, and a 3x3 matrix to input <2> of the **sbencode** function.

B (beeper)

Description

This code indicates a string of bytes, with each byte indicating the sound on/off and duration.

Query Processing

Spaceball returns the remaining sound string.

Command Processing

The **sbin** function always clears wait-for-response flag, and then sends the command. Spaceball starts the internal beeper using the data to determine the length of the on/off sound.

Message Processing

The **sbin** function copies the message to output <9>.

Data Format

The data format contains multiple bytes, each with the following bit format:

```
xx <on/off> <5-bit duration in 1/32 sec>
```

where ***<on/off>*** is defined as

1 = on

0 = off

D (data)

Description

If queried, the data is sent after the minimum pulse increment. If not queried, the data is sent after the maximum pulse increment. Data is sent only if changed and if translation and/or rotation modes are enabled. The length of the pulse increment is specified by the pulse command. The presence of the data and the format of the rotation data is specified by the **mode** command.

Query Processing

The **sbin** function always clears wait-for-response flag, and then sends the query. Data queries can be reset by any other message. For this reason, if a message of any other type is received, a data query is resent. Spaceball sends the message when the minimum pulse increment has expired.

Command Processing

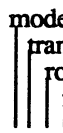
Invalid command.

Message Processing

The **sbin** function processes the message only if the wait-for-response flag is reset. This implies that if any command has been issued which might change the characteristics of the data message, the data messages will be ignored until the response to the command has been received.

Data messages are polled by sending a **'d'** query whenever data messages are enabled and any message is received from Spaceball. The query causes a data message to be returned as soon as manipulation of Spaceball causes the data to become available.

Data Format



"MSSX": see mode command

```
<period> <delta tran vector> <rot data>
```

"MSN":

```
<period> <delta tran vector>
```

"MNSV":

```
<period> <rot vector>
```

"MNSD":

```
<delta rot matrix>
```

"MNSA":

```
<absolute rot matrix (premultiplied)>
```

"MNSa":

```
<absolute rot matrix (postmultiplied)>
```

where:

<period> is a 16-bit unsigned scale factor in microseconds/62.5 for a maximum of 4.096 s. This value is present with translation and rotation vectors.

<tran vector> is three 16-bit signed values to be scaled by the period. It is sent if translation is enabled. For message processing, a three-dimensional vector message is sent to output <4>.

<rot vector> is three 16-bit signed values to be scaled by the period. The values are the angles about the *x*, *y*, *z* axes. For message processing, a three-dimensional vector message sent to output <5>.

<delta rot matrix> is a compact rotation matrix.

<absolute rot matrix> is a compact rotation matrix. For message processing, a 3x3 delta matrix sent to output <7>. The data is saved in a private queue for reset.

E (error)

Description

This code indicates that an internal error has occurred.

Command Processing

Invalid command.

Message Processing

The **sbin** function clears the wait-for-response flag, and then copies the message to output <8>.

Data Format

```
<error code> [ <error code> ... ]
```

where:

<error code> is one or more ASCII characters. Use the **help** command to get more information.

F (feel)

Description

This code is a response multiplier to allow nonlinear output in response to increasing force or torque on Spaceball.

Query Processing

Spaceball returns both translation and rotation **feel** messages.

Command Processing

The **sbIn** function sends command. Spaceball updates the **feel** values and returns both translation and rotation **feel** messages.

Message Processing

The **sbIn** function saves the message in a private queue for reset, and then copies the message to output <9>.

Data Format

```
<feel type- T-tran, R-rot><feel>
```

where:

<feel type> is a printable ASCII character. Values $00 through $3F cause a linear through cubic response.

(Initialized by Spaceball to "T?") ($3F), and "RL") ($4C))

(Initialized by function to "T?") ($3F), and "RL") ($4C))

Notes

feel commands can be created by sending an **'F'** and **'T'** or **'R'**, for translation and rotation, to input <1> of the **sbencode** function, and an integer or real value to input <2> of the **sbencode** function.

H (help)

Description

This code returns a list of all the commands and the length and type of the associated data. It may also be used to return the text of a message describing an error.

Query Processing

The **sbin** function always clears the wait-for-response flag, and then sends a query. Spaceball returns the requested help messages.

Command Processing

Invalid command.

Message Processing

The **sbin** function clears the wait-for-response flag, and then copies the message to output <9>.

Data Format

```
<error code> | "v" | "?" | nothing
```

where:

<error code> is an ASCII character sent with the "E" message.

"v" indicates that Spaceball responds with firmware version and date.

"?" indicates that Spaceball responds with number of invalid packets received since the last ***"h?"*** query and resets the count of invalid packets.

"nothing" indicates that Spaceball responds with a list of all valid commands and formats.

K (keypad)

Description

This code indicates that two bytes containing bits will indicate the pressed and released state of each button.

Query Processing

Spaceball returns the current button status.

Command Processing

Invalid command.

Message Processing

The **sbin** function clears the wait-for-response flag, and then saves the message in a private queue to determine when the button status changes. If a button is pressed, an integer corresponding to the button is sent to output <2>. If a button is released, an integer is sent to output <3>.

Data Format (bits)

```
010<pick><b8><b7><b6><b5>0100<b4><b3><b2><b1>
```

M (mode)

Description

This code specifies the form and type of data sent in the "D" data message. When a program terminates, the transmission of data should be disabled by changing the data mode. When a program starts, it will usually set the data mode and send an identity absolute matrix if using absolute rotation data. The null radius, feel, spin rate, and relative translation sensitivities can also be set if these values differ from the defaults.

Query Processing

Spaceball returns the current status with the associated message.

Command Processing

The **sbIn** function sends the command. Spaceball updates the internal values and returns the associated message.

Message Processing

The **sbIn** function saves the message in a private queue for reset and for a test to determine if data messages are enabled. The message is also copied to output <9>.

Data Format

```
<tran mode> [ <rot mode> [ <rot form> [ <handedness>
[ <pre/post processing (?)>] ] ] ]
```

where:

<tran mode> and ***<rot mode>*** are defined below.

N - no freedom

L - linear freedom

P - planar freedom

S - spatial freedom

X - do not change

<rot form> is defined below.

V - vector

D - delta matrix

A - absolute matrix

X - do not change

<handedness> is defined below.

L - left-handed coordinate system

R - right-handed coordinate system

X - do not change

<pre/post processing> is defined below.

A - post-multiplication

B - pre-multiplication

(Initialized by **@RESET** command to "MNNVLA")

(Initialized by the function to "MNNALB")

Notes

When an application program terminates, it should disable both translation and rotation data so that Spaceball is not left sending data.

N (null radius)

Description

This code indicates the value in internal Spaceball units of the radius within which no changes in Spaceball are to be reported.

Query Processing

Spaceball returns the current status with the associated message.

Command Processing

The **sbin** function sends the command, and Spaceball updates the internal values and returns the associated message.

Message Processing

The **sbin** function saves the message in a private queue for reset.

Data Format

```
<null radius>
```

where:

<null radius> is a printable ASCII character.

(Initialized to - " ' " ("$60"))

Notes

The **null radius** can be used to make Spaceball respond to less force. However, decreasing the **null radius** makes it more likely that Spaceball will send data when it is not being manipulated.

null radius commands can be created by sending an **'N'** to input <1> of the **sbencode** function and an integer or real value to input <2> of the **sbencode** function.

O (orientation)

Description

This code changes the viewing position of Spaceball.

Query Processing

Spaceball returns the current status with the associated message.

Command Processing

The **sbin** function sends the command, and Spaceball updates the internal values and returns the associated message.

Message Processing

The **sbin** function saves the message in a private queue for reset and copies the message to output <9>. A **Qpacket** with code **'O'** and the decoded matrix is sent to output <10>.

Data Format

```
<Compact Rotation Matrix>
```

(Initialized to "$000000000000")

Notes

orientation commands can be created by sending an **'O'** to input <1> of the **sbencode** function and a 3x3 matrix to input <2> of the **sbencode** function.

P (pulse)

Description

This code sets the minimum length of time to wait before sending a data message after Spaceball is polled, or the maximum length of time to wait before sending a message if Spaceball is not polled.

Query Processing

Spaceball returns the current status with the associated message.

Command Processing

The **sbIn** function sends the command, and Spaceball updates the internal values and returns the associated message.

Message Processing

The **sbIn** function saves the message in a private queue for reset.

Data Format

```
<max period> [ <min period> ]
```

where:

<max period> and ***<min period>*** are two printable ASCII characters, the lowest 6 bits of each combining to form a 12-bit unsigned integer period in milliseconds, for a max of 4.095 seconds.

(Initialized by Spaceball to "PW\@h" ("$575C4068") Max- 1500, Min- 40)

(Initialized by Function to "P??@A" ("$3F3F4041") Max- 4095, Min- 1)

Notes

The **sbIn** function initializes the minimum value to be as small as possible, so that Spaceball will return data as fast as possible when it is polled. Since Spaceball is always polled by the **sbIn** function, the maximum value is not really used. However, to prevent extraneous data messages in the case where function networks are running slower than the maximum pulse increment, the maximum increment is initialized to the highest value available.

This value is initialized by the **sbIn** function and should not be updated by the user.

R (rotation mode)

Description

This code controls the local/parent (object/world) mode of the ball system, the freedom vector system, and the output system.

Query Processing

Spaceball returns the current status with the associated message.

Command Processing

The **sbIn** function sends the command, and Spaceball updates the internal values, and returns the associated message.

Message Processing

The **sbIn** function saves the message in a private queue for reset and copies the message to output <9>. A **Qpacket** with code **'R'** and the decoded vector is sent to output <10>.

Data Format

```
<ball sys> <freedom vector sys> <output sys>
[ <rot freedom vector> ]
```

where:

<ball sys>, ***<freedom vector sys>***, and ***<output sys>*** are defined below.

L - local

P - parent

X - do not change

<rot freedom vector> is three 16-bit signed integers which form a vector with a length of 2**14.

(Initialized to "PPP""$400000000000")

Notes

rotation freedom vector commands can be created by sending an **'R'** and the ball, freedom vector, and output system values to input <1> of the **sbencode** function, and a three-dimensional vector to input <2> of the **sbencode** function. The **sbencode** function converts the vector into a proportional unit vector.

S (spin rate)

Description

This code indicates the exponent and mantissa of the rotation matrix sensitivity multiplier.

Query Processing

Spaceball returns the current status with the associated message.

Command Processing

The **sbin** function sends the command, and Spaceball updates the internal values and returns the associated message.

Message Processing

The **sbin** function saves the message in a private queue for reset and copies the message to output <9>.

Data Format

```
<spin rate mantissa> <spin rate exponent>
```

where:

<spin rate exponent> and ***<spin rate mantissa>*** are printable ASCII characters.

(Initialized to "pK'"$704B")

Notes

spin rate commands can be created by sending an **'S'** to input <1> of the **sbencode** function and a two-dimensional vector to input <2> of the **sbencode** function. The values of the two-dimensional vector are (***mantissa,exponent***). The **sbencode** function converts each of the components of the vector into a single printable ASCII character.

The mantissa values are converted by Spaceball to be `(n MOD 32) + 32)`. For example,

If a '1' is sent, (1 MOD 32) + 32 = 33
If a '33' is sent, (33 MOD 32) + 32 = 33

The exponent values are converted by Spaceball to be `(n MOD 16)`. For example,

If a '1' is sent, `(1 MOD 16) = 1`
If a '17' is sent, (17 MOD 16) = 1
If a '33' is sent, (33 MOD 16) = 1

spin rate applies to delta and absolute matrix forms of rotation data, and does not apply to translation or rotation vector data.

T (translation mode)

Description

This code controls the local/parent (object/world) mode of the ball system, the freedom vector system, and the output system.

Query Processing

Spaceball returns the current status with the associated message.

Command Processing

The **sbIn** function sends the command, and Spaceball updates the internal values and returns the associated message.

Message Processing

The **sbIn** function saves the message in a private queue for reset and copies the message to output <9>. A **Qpacket** with code **'T'** and the decoded vector is sent to output <10>.

Data Format

```
<ball sys> <freedom vector sys> <output sys>
[ <tran freedom vector> ]
```

where:

<ball sys>, ***<freedom vector sys>***, and ***<output sys>*** are defined below.

L - local

P - parent

X - do not change

<tran freedom vector> is three 16-bit signed integers which form a vector with a length of 2**14.

(Initialized to "PPP""$400000000000")

Notes

translation freedom vector commands can be created by sending a **'T'** and the ball, freedom vector, and output system values to input <1> of the **sbencode** function, and a three-dimensional vector to input <2> of the **sbencode** function. The **sbencode** function converts the vector into a proportional unit vector.

X (XYZ relative translation sensitivities)

Description

This code sets the relative translation sensitivity of the *x,y,z* axes.

Query Processing

Spaceball returns the current status with the associated message.

Command Processing

The **sbin** function sends the command, and Spaceball updates the internal values and returns the associated message.

Message Processing

The **sbin** function saves the message in a private queue for reset and copies the message to output <9>. A **Qpacket** with code **'X'** and the decoded vector is sent to output <10>.

Data Format

```
<X-sensitivity><Y-sensitivity><Z-sensitivity>
```

where:

<X-sensitivity>, ***<Y-sensitivity>***, and ***<Z-sensitivity>*** are 16-bit integers which form a vector with a length of 2**14.

(Initialized to "$400020002000")

Notes

relative translation sensitivities commands can be created by sending an **'X'** to input <1> of the **sbencode** function, and a three-dimensional vector to input <2> of the **sbencode** function. The **sbencode** function converts the three-dimensional vector into a proportional unit vector.

Z (zero)

Description

This code causes the current state of Spaceball to be considered the initial (zero) state.

Query Processing

Spaceball returns the current status with the associated message.

Command Processing

The **sbin** function always clears wait-for-response flag, and then sends the command. Spaceball updates the internal values and returns the associated message.

Message Processing

The **sbin** function copies the message to output <9>.

Data Format

Undefined. Spaceball can be queried for the current **zero** setting and the setting can be sent as a command later to reset it to the previous value.

Notes

An application should zero Spaceball only when no objects are being displayed to reduce the likelihood of manipulating Spaceball when zeroing occurs. The sending of data should be controlled by using the **mode** command, and the **zero** function assigned to a keypad button.

A. ES/PSX Example Program

Installation Instructions

1) Include the following path in the ***PSX_PATH*** environment variable: **/usr/lib/psx/demo.**

2) Run ES/PSX.

3) Put the keyboard into command mode by pressing the left mouse button when the cursor is in the "keyboard mode" portion of the **psx** menu.

4) At the `@@` prompt, load the example program by entering the following:

```
send 'gencar.dat' to <1>readascii;
send 'sbdemo.dat' to <1>readascii;
```

5) After the example program has been loaded, information boxes, notes, and a wire-frame car model will be displayed on the monitor.

6) Spaceball can now be used to control rotations, translations, color hue and saturation, zeroing Spaceball, and resetting the monitor.

7) To end the example program, enter the following at the `@@` prompt:

```
init;
```

Note: Spaceball can be left in a state where it is continuously sending data. Disable it by disconnecting the RS-232-C cable or the power cable.

Files

The following two files are included as the example program:

- **gencar.dat**

  This binary file contains the vector list and display structure for the wire-frame car model.

- **sbdemo.dat**

  This ASCII file contains the demonstration network.

Function Network

The demonstration network uses outputs <2>, <3>, <4>, and <7> of the **sbin** function to control both the motion and color aspect of the orthographic, wire-frame model.

Output <2> of the **sbin** function is connected to the **tran_route, buttons** and **color_reset** functions. The output is an integer corresponding to the Spaceball button pressed (1 through 8, and 9 for the pick button). The integer selects the output through which to route the **tran_route** and **buttons** function input. The integer also triggers the **color_reset** function to set all menu selections to green.

Output <3> of the **sbin** function is connected to the **button_rel** function. The output is an integer corresponding to the Spaceball button pressed (1 through 8, and 9 for the pick button). When the Spaceball button is released, the corresponding menu selection is set to green.

Output <4> of the **sbin** function (delta translation vectors) is routed through the **tran_route** function. If button 1, 2, 4, 5, 6, 8, or 9 is pressed, the delta translation vectors are sent through the **tranmult** function and then to the **tran_acc** and **zcoord** functions.

The **zcoord** function removes all but the *z*-axis translations and inputs them to the **negit, scaleit,** and **scalemat** functions. These functions are used to enhance the *z*-translation by scaling the model. The object appears smaller as it recedes in the *z*-direction.

Output <3> of the **tran_route** function is used when button 3 is pressed for the color mode. Delta translation vectors are split into their *x* and *z* components through the **color_parts** function and then input to the **hue_acc** and **sat_acc** functions. The numeric output of these functions controls the hue and saturation of the model through the Picture.Col node. These functions also set the minimum and maximum values for hue and saturation. The output of the **hue_acc** and **sat_acc** functions is also input into the **color_vec** function. The **color_vec** function output controls the dot in the color box display at the top of the screen through the Color_Box.B.Trn node of the display structure.

Output <7> of the **tran_route** function is used to control the intensity of the model in a manner similar to the color mode. Only *z*-translations are used from the **inten_parts** function are input to the **inten_acc** function. The output of the **inten_acc** function is input to the **inten_vec** function which creates 2D vectors with a constant X of zero. These vectors are then input to the Display Intensity Box node of the display structure (Color_Box.I.Trn) and the Model node of the display structure (Picture.Int).

The **buttons** function is used as a trigger for the following twelve functions:

1 - **sbtrnon** turn translations on
2 - **sbroton** turn rotations on
3 - **docolor** go to color mode to control hue and saturation
4 - **sbsix** turn translations and rotations on, 6-axis mode
5 - **sbtrnoff** turn translations off
6 - **sbrotoff** turn rotations off
7 - **doint** go to intensity mode to control intensity
8 - **sbrezero** rezero Spaceball
9 - **hometrn** reset the display to home position, activate **sbsix**
homerota
homerotb
homescl

When they are triggered, the first eight functions send an ASCII string command to the **sbout** function to execute the desired operation. Button 9, the pick button, causes home translation vectors to be sent to Picture and a home identity absolute rotation matrix to be sent to Spaceball through the **sbencode** function. The **homesync** function is used to coordinate the arrival of the two required inputs to the **sbencode** function.

The **button_rel** function connects to the nine functions, **chg_colora** through **chg_colori**. The **color_reset** function is also connected to these nine functions. When a button is pressed, a Boolean true is sent to all **chg_color** functions, causing 120 (red) to be sent to all menu choice color nodes through output <2> of the **sbin** function, **color_reset**, and **chg_colora** through **chg_colori**. When a button is released, a Boolean false is sent from the **button_rel** function to only one of the **chg_color** functions, causing 240 (green) to be sent to the appropriate menu selection color node.

sbdemo.dat Program Listing

```
{ sbdemo.dat:SPACEBALL DEMO NETWORK                                    }
{                                                                      }
{ Version 1.3     February 1, 1991                                     }
{                                                                      }
{ Copyright (c)   Evans & Sutherland                                   }
{                                                                      }
{ Create all of the functions required for the network                 }
TRAN_ACC     := f:accumulate;
ZCOORD       := f:parts;
NEGIT        := f:csub;
SCALEIT      := f:accumulate;
SCALEMAT     := f:scale;
TRANMULT     := f:mulc;
DOCOLOR      := f:constant;
TRAN_ROUTE   := f:croute(9);
COLOR_PARTS  := f:parts;
HUE_ACC      := f:accumulate;
SAT_ACC      := f:accumulate;
COLOR_VEC    := f:vec;
DOINT        := f:constant;
INTEN_PARTS  := f:parts;
INTEN_ACC    := f:accumulate;
INTEN_VEC    := f:cvec;
BUTTON       := f:routec(9);
SBTRNON      := f:constant;
SBROTON      := f:constant;
SBSIX        := f:constant;
SBTRNOFF     := f:constant;
SBROTOFF     := f:constant;
SBREZERO     := f:constant;
HOMETRN      := f:constant;
HOMESCL      := f:constant;
HOMEROTA     := f:constant;
HOMEROTB     := f:constant;
HOMESYNC     := f:sync(2);
SCLTRIG      := f:constant;
BUTTON_REL   := f:routec(9);
COLOR_RESET  := f:constant;
CHG_COLORA   := f:boolean_choose;
CHG_COLORB   := f:boolean_choose;
CHG_COLORC   := f:boolean_choose;
CHG_COLORD   := f:boolean_choose;
CHG_COLORE   := f:boolean_choose;
```

```
CHG_COLORF  := f:boolean_choose;
CHG_COLORG  := f:boolean_choose;
CHG_COLORH  := f:boolean_choose;
CHG_COLORI  := f:boolean_choose;
{ Disconnect outputs so that any unused outputs do not      }
{ generate a warning message                                }
disconnect TRAN_ROUTE:all;
disconnect ZCOORD:all;
disconnect INTEN_PARTS:all;
disconnect COLOR_PARTS:all;
{============================================================}
{ Create the display node for the wire frame model (gencar)  }
PICTURE := begin_structure
TRN := translate 0,0,0;
ROT := rotate x 0;
SCL := scale 1;
INT := set intensity on .01:.5;
COL := set color 180,1;
instance gencar;
end_structure;  { end PICTURE }

{============================================================}
{ Create the display node for the demo menu                 }
COLOR_BOX := begin_structure
begin_structure
set color 120,0.85;
character scale 0.05;
label -.86,.92 'SPACEBALL ORTHOGRAPHIC DEMONSTRATION';
end_structure;
B := begin_structure
set color 120,0.85;
viewport horizontal = -.9:-.6 vertical = .75:.89 ;
translate -1,-1;
scale .0055555555,1.999,1.999;
TRN := vector dots n=1 180,1.0;
set intensity on .2:.2;
vector item n=5  p 0,0   l 360,0   l 360,1   l 0,1   l 0,0;
end_structure;  { end COLOR_BOX.B }
LABELA := begin_structure
COLORA := set color 120,1;
character scale 0.02;
label -.22,.85 'Turn tran on';
end_structure;  { end COLOR_BOX.LABELA }
LABELB := begin_structure
```

```
COLORB := set color 120,1;
character scale 0.02;
label .08,.85 'Turn rot on';
end_structure;  { end COLOR_BOX.LABELB }
LABELC := begin_structure
COLORC := set color 120,1;
character scale 0.02;
label -.78,.70 'Hue';
label -.96,.85 'S';
label -.96,.82 'a';
label -.96,.79 't';
label .38,.85 'Color mode';
end_structure;  { end COLOR_BOX.LABELC }
LABELD := begin_structure
COLORD := set color 120,1;
character scale 0.02;
label .68,.85 '6-axis mode';
end_structure;  { end COLOR_BOX.LABELD }
LABELE := begin_structure
COLORE := set color 120,1;
character scale 0.02;
label -.22,.8 'Turn tran off';
end_structure;  { end COLOR_BOX.LABELE }
LABELF := begin_structure
COLORF := set color 120,1;
character scale 0.02;
label .08,.8 'Turn rot off';
end_structure;  { end COLOR_BOX.LABELF }
LABELG := begin_structure
COLORG := set color 120,1;
character scale 0.02;
label -.47,.7 'Intensity';
label .38,.8 'Intensity mode';
end_structure;  { end COLOR_BOX.LABELG }
LABELH := begin_structure
COLORH := set color 120,1;
character scale 0.02;
label .68,.8 'Rezero SPACEBALL';
end_structure;  { end COLOR_BOX.LABELH }
LABELI := begin_structure
COLORI := set color 120,1;
character scale 0.02;
label -.1,.75 'Pick Button : Reset display, go to 6-axis mode';
end_structure;  { end COLOR_BOX.LABELI }
```

```
NUM_LABELS := begin_structure
set color 120,0.85;
character scale 0.02;
label -.905,.71 '0';
label -.62,.71 '360';
label -.93,.75 '0';
label -.93,.88 '1';
end_structure;  { end COLOR_BOX.NUMLABELS }
I := begin_structure
set color 120,0.85;
viewport horizontal = -.4:-.38 vertical = .75:.89;
translate 0,-1,0;
scale 0.999,1.999,0.999;
begin_structure
set intensity on .2:.2;
vector item n=5   p -1,0   l 1,0   l 1,1   l -1,1   l -1,0;
end_structure;
TRN := translate 0,.5;
vector item n=2   p-1,0   l 1,0;
end_structure;  { end COLOR_BOX.I }
end_structure;  { end COLOR_BOX }

{============================================================}
{ Build the network connections and set all initial and      }
{ constant values                                            }
{ Connections to the network from SPACEBALL                  }
connect SBIN<2>:<1>BUTTON;
connect SBIN<2>:<1>COLOR_RESET;
connect SBIN<3>:<1>BUTTON_REL;
connect SBIN<2>:<1>TRAN_ROUTE;
connect SBIN<4>:<2>TRAN_ROUTE;
{ Absolute rotation matrices go directly to the model        }
{ rotate node                                                }
connect SBIN<7>:<1>PICTURE.ROT;
{ Set up the constants necessary to change the color         }
{ of the active menu choice from red to green                }
send false to <2>BUTTON_REL;
send true to <2>COLOR_RESET;
send 120 to <2>CHG_COLORA;
send 240 to <3>CHG_COLORA;
send 120 to <2>CHG_COLORB;
send 240 to <3>CHG_COLORB;
send 120 to <2>CHG_COLORC;
send 240 to <3>CHG_COLORC;
send 120 to <2>CHG_COLORD;
```

```
send 240 to <3>CHG_COLORD;
send 120 to <2>CHG_COLORE;
send 240 to <3>CHG_COLORE;
send 120 to <2>CHG_COLORF;
send 240 to <3>CHG_COLORF;
send 120 to <2>CHG_COLORG;
send 240 to <3>CHG_COLORG;
send 120 to <2>CHG_COLORH;
send 240 to <3>CHG_COLORH;
send 120 to <2>CHG_COLORI;
send 240 to <3>CHG_COLORI;
{ Connections to reset all menu choices to red when any button }
{ is pressed                                                    }
connect COLOR_RESET<1>:<1>CHG_COLORA;
connect COLOR_RESET<1>:<1>CHG_COLORB;
connect COLOR_RESET<1>:<1>CHG_COLORC;
connect COLOR_RESET<1>:<1>CHG_COLORD;
connect COLOR_RESET<1>:<1>CHG_COLORE;
connect COLOR_RESET<1>:<1>CHG_COLORF;
connect COLOR_RESET<1>:<1>CHG_COLORG;
connect COLOR_RESET<1>:<1>CHG_COLORH;
connect COLOR_RESET<1>:<1>CHG_COLORI;
{ Connections to set the menu choice selected to green when }
{ the button is released                                    }
connect BUTTON_REL<1>:<1>CHG_COLORA;
connect BUTTON_REL<2>:<1>CHG_COLORB;
connect BUTTON_REL<3>:<1>CHG_COLORC;
connect BUTTON_REL<4>:<1>CHG_COLORD;
connect BUTTON_REL<5>:<1>CHG_COLORE;
connect BUTTON_REL<6>:<1>CHG_COLORF;
connect BUTTON_REL<7>:<1>CHG_COLORG;
connect BUTTON_REL<8>:<1>CHG_COLORH;
connect BUTTON_REL<9>:<1>CHG_COLORI;
CONNECT CHG_COLORA<1>:<1>COLOR_BOX.LABELA.COLORA;
CONNECT CHG_COLORB<1>:<1>COLOR_BOX.LABELB.COLORB;
CONNECT CHG_COLORC<1>:<1>COLOR_BOX.LABELC.COLORC;
CONNECT CHG_COLORD<1>:<1>COLOR_BOX.LABELD.COLORD;
CONNECT CHG_COLORE<1>:<1>COLOR_BOX.LABELE.COLORE;
CONNECT CHG_COLORF<1>:<1>COLOR_BOX.LABELF.COLORF;
CONNECT CHG_COLORG<1>:<1>COLOR_BOX.LABELG.COLORG;
CONNECT CHG_COLORH<1>:<1>COLOR_BOX.LABELH.COLORH;
CONNECT CHG_COLORI<1>:<1>COLOR_BOX.LABELI.COLORI;
{ Route the delta translation vectors according to which   }
{ button is pressed                                        }
```

```
connect TRAN_ROUTE<1>:<1>TRANMULT;
connect TRAN_ROUTE<2>:<1>TRANMULT;
connect TRAN_ROUTE<3>:<1>COLOR_PARTS;
connect TRAN_ROUTE<4>:<1>TRANMULT;
connect TRAN_ROUTE<5>:<1>TRANMULT;
connect TRAN_ROUTE<6>:<1>TRANMULT;
connect TRAN_ROUTE<7>:<1>INTEN_PARTS;
connect TRAN_ROUTE<8>:<1>TRANMULT;
connect TRAN_ROUTE<9>:<1>TRANMULT;
{ Connections to add a constant multiplier to translation  }
{ vectors and to send translation vectors to an accumulator }
{ for X, Y and to a parts function for Z so as to apply     }
{ scaling to the Z translation                              }
connect TRANMULT<1>:<1>TRAN_ACC;
connect TRANMULT<1>:<1>ZCOORD;
send 20 to <2>TRANMULT;
{ For initializing and controlling the menu and model       }
connect TRAN_ACC<1>:<1>PICTURE.TRN;
send v(0,0,0) to <2>TRAN_ACC;
send 1 to <4>TRAN_ACC;
send v(1,1,0) to <5>TRAN_ACC;
send v(-1,-1,0) to <6>TRAN_ACC;
{ Connect and initialize Z translation and scale            }
connect ZCOORD<3>:<2>NEGIT;
connect NEGIT<1>:<1>SCALEIT;
send 0 to <1>NEGIT;
connect SCALEIT<1>:<4>SCALEIT;
connect SCALEIT<1>:<1>SCALEMAT;
send 1 to <2>SCALEIT;
send 5 to <5>SCALEIT;
send 0.001 to <6>SCALEIT;
connect SCALEMAT<1>:<1>PICTURE.SCL;
{ Connect and initialize BUTTON to perform the various      }
{ SPACEBALL control functions                               }
send true to <2>BUTTON;
connect BUTTON<1>:<1>SBTRNON;
connect SBTRNON<1>:<1>SBOUT;
send 'MSX' to <2>SBTRNON;
connect BUTTON<2>:<1>SBROTON;
connect SBROTON<1>:<1>SBOUT;
send 'MXS' to <2>SBROTON;
connect BUTTON<3>:<1>DOCOLOR;
connect DOCOLOR<1>:<1>SBOUT;
send 'MSN' to <2>DOCOLOR;
```

```
connect BUTTON<4>:<1>SBSIX;
connect SBSIX<1>:<1>SBOUT;
send 'MSS' to <2>SBSIX;
connect BUTTON<5>:<1>SBTRNOFF;
connect SBTRNOFF<1>:<1>SBOUT;
send 'MNX' to <2>SBTRNOFF;
connect BUTTON<6>:<1>SBROTOFF;
connect SBROTOFF<1>:<1>SBOUT;
send 'MXN' to <2>SBROTOFF;
connect BUTTON<7>:<1>DOINT;
connect DOINT<1>:<1>SBOUT;
send 'MSN' to <2>DOINT;
connect BUTTON<8>:<1>SBREZERO;
connect SBREZERO<1>:<1>SBOUT;
send 'Z' to <2>SBREZERO;
connect BUTTON<9>:<1>HOMETRN;
connect BUTTON<9>:<1>HOMESCL;
connect BUTTON<9>:<1>HOMEROTA;
connect BUTTON<9>:<1>HOMEROTB;
connect BUTTON<9>:<1>SBSIX;
{ Connections for controlling hue, saturation and intensity of}
{ model and also to control the menu display                   }
connect COLOR_PARTS<1>:<1>HUE_ACC;
connect COLOR_PARTS<3>:<1>SAT_ACC;
connect HUE_ACC<1>:<1>COLOR_VEC;
connect HUE_ACC<1>:<1>PICTURE.COL;
connect SAT_ACC<1>:<2>COLOR_VEC;
connect SAT_ACC<1>:<2>PICTURE.COL;
connect COLOR_VEC<1>:<1>COLOR_BOX.B.TRN;
send 200 to <4>HUE_ACC;
send 359 to <5>HUE_ACC;
send 1 to <6>HUE_ACC;
send 10 to <4>SAT_ACC;
send 1 to <5>SAT_ACC;
send 0 to <6>SAT_ACC;
send 180 to <2>HUE_ACC;
send 1 to <2>SAT_ACC;
connect INTEN_PARTS<3>:<1>INTEN_ACC;
connect INTEN_ACC<1>:<2>INTEN_VEC;
connect INTEN_VEC<1>:<1>COLOR_BOX.I.TRN;
connect INTEN_VEC<1>:<2>PICTURE.INT;
send 0.5 to <2>INTEN_ACC;
send 10 to <4>INTEN_ACC;
send 1.0 to <5>INTEN_ACC;
```

```
send 0.0 to <6>INTEN_ACC;
send 0.0 to <1>INTEN_VEC;
{ Connections and constant values to reset the model to its    }
{ initial screen location                                      }
connect HOMETRN<1>:<1>TRAN_ACC;
connect HOMETRN<1>:<2>TRAN_ACC;
send v(0,0,0) to <2>HOMETRN;
connect HOMESCL<1>:<2>SCALEIT;
connect HOMESCL<1>:<1>SCLTRIG;
send 1 to <2>HOMESCL;
connect SCLTRIG<1>:<1>SCALEIT;
send 0 to <2>SCLTRIG;
connect HOMEROTA<1>:<1>HOMESYNC;
send 'A' to <2>HOMEROTA;
connect HOMEROTB<1>:<2>HOMESYNC;
send m3d(1,0,0 0,1,0 0,0,1) to <2>HOMEROTB;
connect HOMESYNC<1>:<1>SBENCODE;
connect HOMESYNC<2>:<2>SBENCODE;
{ decrease the null radius to improve response }
send 'N?' to <1>SBOUT;
{ change the spin rate to improve rotational performance    }
send 'SpF' to <1>SBOUT;
{ make rotations car oriented rather than world oriented    }
send 'RLXX' to <1>SBOUT;
{ Give ES/PSX time to do all of the above before continuing }
give_up_cpu;
give_up_cpu;
give_up_cpu;
give_up_cpu;
give_up_cpu;
init display;
display COLOR_BOX;
display PICTURE;
{ Make sure Spaceball is zeroed and the model is reset to   }
{ initial location                                          }
send fix(8) to <1>BUTTON;
send fix(9) to <1>BUTTON;
send fix(8) to <1>BUTTON;
send fix(9) to <1>BUTTON;
```