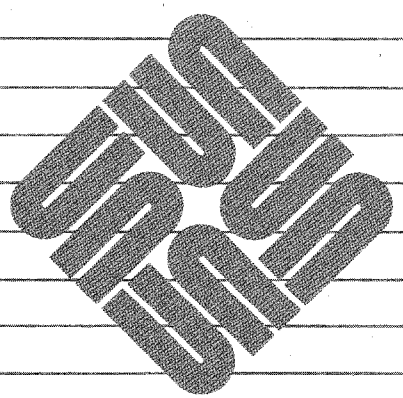
sun®
microsystems

SunCore® Reference Manual



Part Number: 800-1787-10
Revision A, of 9 May 1988

Contents

Tables

Figures

Preface

This document describes *SunCore*, an implementation of the *ACM SIGGRAPH Core System* by Sun Microsystems, Inc. *SunCore* conforms to level 3C (dynamic output with 3D scaling, rotation and translation) of the Core specification for output primitives, and to level 2 (complete input) for input primitives. Appendix A summarizes the differences between *SunCore* and ACM SIGGRAPH Core System.

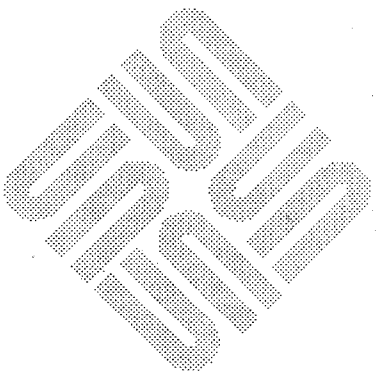
Controlling Document

The following document was used in interpreting the ACM SIGGRAPH Core System:

[1] *Status Report of the Graphics Standards Planning Committee.* Computer Graphics. Volume 13, Number 3, August 1979.

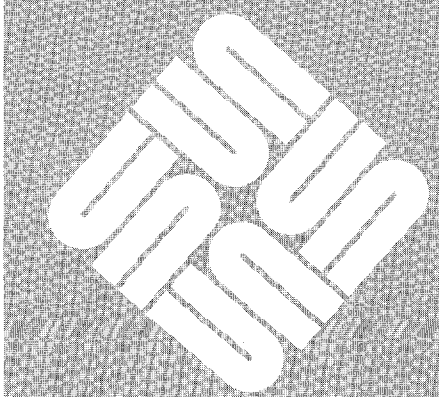
Audience

The intended reader of this document is an applications programmer who is familiar with interactive computer graphics and the C programming language. This manual contains several example programs that can be used as templates for larger *SunCore* applications.



1

Introduction



1

Introduction

SunCore is a comprehensive package of engineering graphics software providing the underlying support for interactive graphics applications programs. It is based on the ACM Core System, a graphics standard designed for 3D interactive graphics.

SunCore provides extensions to the ACM Core System. These include textured polygon fill algorithms, raster primitives, RasterOp attributes, shaded surface polygon rendering, and hidden surface elimination.

SunCore supports both the high resolution monochrome bitmap displays and the Sun color displays. Device-dependent functions support all these displays under *SunCore*. *SunCore* can also be used in conjunction with the Sun Graphics Processor and Graphics Buffer options.

Note that this manual is a *reference manual* for the *SunCore* graphics package. It is not a tutorial for the programmer without knowledge of graphics principles. It assumes that the reader is familiar with the concepts of graphics, and has some familiarity with the ACM Core specification. Those who are new to graphics should consult one of the publications listed in Section 1.7.

1.1. Where to Start

If you are an applications programmer who is familiar with the ACM Core specification, but are new to *SunCore*, it is recommended that you read Appendix A in order to become familiar with the areas where *SunCore* deviates from and provides extensions to the ACM Core specification.

Note that *SunCore* supports the ACM Core output level 3C, that is, dynamic output is supported, including 2 and 3D translation, scaling and rotation. *SunCore* supports the ACM Core input level 2, that is, synchronous input, including the PICK device. *SunCore* supports dimension level 2, that is, 3D operations.

1.2. Overview and Terminology

The objective of a graphics application program is drawing pictures and text on some display device, be it an ephemeral display device such as TV monitor or terminal, or a hard copy device such as a plotter or printer.

There is a need for a device-independent way of representing graphics images in the computer, and having a collection of software functions map the device-independent representations into the physical representations that the output device can handle. *SunCore* is an implementation of one of the "standard" packages of graphics software that have appeared recently. This section introduces

some of the terminology of *SunCore*. This terminology is used throughout this manual. It is somewhat easier to describe the terminology from the point of view of the physical device working backwards to the application program, rather than starting at the software and working out to the device.

There are two quite distinct points of view for looking at a system running a graphics application:

- The physical device (monitor, printer, and so on) on which the final pictures appear, and
- The internal world which the programmer uses to describe the pictures, and which (because of *SunCore*) is independent of the physical device.

A *view surface* is a physical surface on which the final picture appears.

There are two interdependent sets of coordinate systems in use in the graphics package:

World Coordinates
: is a coordinate system which is device-independent. The applications programmer constructs all graphical objects in terms of world coordinates.

Normalized Device Coordinates
: (often abbreviated as NDC) is a fixed coordinate system which is independent of physical output devices. World coordinates are transformed to NDC space for clipping and other operations. Each physical output device driver then transforms from NDC space to the physical device coordinates for each view surface.

A *viewport* is a region of NDC space which the programmer selects and on which the pictures will appear.

It is the job of the viewing transformations to perform the correct mapping between world coordinates and NDC space.

A *window* is a region defined in world coordinates within which the images that the application program defines appear. The selection of the coordinates for the window are arbitrary — the graphics package maps the window into the viewport.

In 3D, the transformation from the window to the viewport is a relatively straightforward process. In 3D, another level of complexity is introduced with the notion of a *view plane* which is positioned arbitrarily in world coordinates.

An *output primitive*, or often just a *primitive*, is a part of a picture (such as a line or a character string). The appearance of primitives (such as solid or dotted lines) is determined by *primitive attributes*. A *primitive attribute* is a general characteristic of an output primitive, and affects the appearance of that primitive. Examples of primitive attributes are color, linestyle, and linewidth.

Each output primitive may be assigned a name, called the *pick-id*, which is used to identify that primitive when an input operation (such as pointing at the primitive with the mouse) is applied.

The *current position* is a *SunCore* system value that defines the current location for drawing. At startup time, the current position is set to the origin of the world coordinate system. Functions that create output primitives (move, line, and so on) can alter the current position.

Output primitives are collected together in *segments*. A segment defines an *image* which is a part of the picture on a view surface.

Segments are divided into two classes, namely: *temporary* and *retained*. A retained segment has a name, and can have segment attributes associated with it. A temporary segment is nameless, and furthermore, the image that a temporary segment defines only remains visible as long as information is only being added to the view surface. As soon as a new frame action (one which repaints view surface) occurs, the temporary segment's image disappears from the view surface.

Each retained segment has one static attribute, its image transformation type. The value of this attribute can be *none, translatable , or transformable. Translatable and transformable retained segments can be translated or transformed in either 2 or 3D.*

Segments also have *dynamic attributes*. The *visibility* and *highlighting* attributes control the appearance of the image. The *detectability* attribute determines if the segment can be detected by the pick device. Dynamic attributes for translatable and transformable segments include the segment's image transformation. Depending on the image transformation type, the image transformation may contain translation, rotation, and scaling components.

A *viewing operation* is an operation that maps positions in world coordinates to positions in NDC space. The viewing operation also determines the portion of the world coordinate space that is visible if window clipping or depth clipping is enabled.

The applications program can obtain user interaction by means of *input primitives*, which provide facilities for pointing at objects, entering data from the keyboard, and causing events.

Basics of Drawing Pictures

The general sequence of actions that an application program goes through to create a picture on a device is this:

1. *Initialize SunCore.*
2. *Initialize a view surface* upon which the picture will be drawn.
3. *Select a view surface* upon which the picture will be drawn.
4. *Specify the viewing operation parameters* (sizes of windows in world coordinates, size of viewport, and so on).
5. *Set* an image transformation type.
6. *Create a segment.* The created segment becomes the currently open segment until it is closed.
7. *Set attributes* for the segment, if required.

8. *Draw objects* in the segment using output primitives.
9. *Close the segment.*
10. Repeat steps 4 through 9 as often as required, for as many segments as needed to build the picture.
11. *Apply image transformations* (translation, scaling, and rotation) to a given segment, to achieve the required picture on the display device.
12. *Deselect the view surface.*
13. *Terminate SunCore.*

In providing the application programmer with the capabilities needed to draw pictures, *SunCore* breaks the interface into six functional areas:

Control
: directs the major actions of *SunCore*, such as startup, shutdown, selection and deselection of view surfaces, and so on.

Segments
: control the creation, closing, and removal of segments. Segments are then used to collect sets of:

Output Functions
: also known as output primitives, which describe the drawing of lines and line sequences, shaded regions, text, and markers.

Attributes
: control the way in which output primitives actually appear in the final image (solid or dotted lines, for instance).

Transformations
: control the major appearances of pictures, such as orientation (rotation), scaling, and translation. Transformations also control projection type and clipping.

Input Functions
: handle the interaction with the user via the keyboard and the mouse.

1.3. Getting Started With SunCore

This section provides an example of a *SunCore* application program. The `glass.c` program draws a martini glass on the screen. This program demonstrates the use of:

- Creating a temporary segment (see *Segmentation and Naming*),
- Moving to an absolute position (see *Output Primitives*),
- Using the polyline drawing functions (see *Output Primitives*),
- Using the absolute line drawing functions (see *Output Primitives*),

```
#include <usercore.h>

static float glassdx[] = { -10.0,9.0,0.0,-14.0,30.0,
                           -14.0,0.0,9.0,-10.0 };
static float glassdy[] = { 0.0,1.0,19.0,15.0,0.0,
                           -15.0,-19.0,-1.0,0.0 };
int pixwindd();
struct vwsurf vwsurf = DEFAULT_VWSURF(pixwindd);

main()
{
    initialize_core(BASIC, NOINPUT, TWOD);
    initialize_view_surface(&vwsurf, FALSE);
    select_view_surface(&vwsurf);
    set_viewport_2(0.125, 0.875, 0.125, 0.75);
    set_window(-50.0, 50.0, -10.0, 80.0);

    create_temporary_segment();
        move_abs_2(0.0, 0.0);
        polyline_rel_2(glassdx, glassdy, 9);
        move_rel_2(-12.0, 33.0);
        line_rel_2(24.0, 0.0);
    close_temporary_segment();

    sleep(10);

    deselect_view_surface(&vwsurf);
    terminate_core();
}
```

Figure 1-1 *Simple Example Program*

`glass.c` can be compiled with the following command line:

```
% cc glass.c -fswitch -o glass -lcore -lsunwindow -lpixrect -lm
```

In the command line above, the options:

`-fswitch`	causes the compiler to take advantage of floating point hardware if it is available. Otherwise, the compiler will emulate this floating point support with software. (For more information on floating point options, see Appendix F).
`-lcore`	selects the *SunCore* run-time library from `/usr/lib/libcore.a`,
`-lsunwindow`	selects the window system library,
`-lpixrect`	selects the pixrect library,
`-lm`	selects the correct math library.

When the compilation is complete, the final program is in the file `glass` and may be run by typing its name.

This example uses the some but not all of *SunCore*'s capabilities. Appendix I contains an example that illustrates other areas of the *SunCore* graphics package.

1.4. The SunCore Lint Library

SunCore provides a `lint(1)` library which provides type checking beyond the capabilities of the C compiler. For example, you could use the *SunCore* `lint(1)` library to check a program called `glass.c` with command like this:

```
% lint glass.c -lcore
```

Note that the error messages that `lint(1)` generates are mostly warnings, and may not necessarily have any effect on the operation of the program. For a detailed explanation of `lint(1)`, see the `lint(1)` chapter in the *Programming Tools* manual.

1.5. The Coordinate Systems

Applications programs which draw pictures using *SunCore* communicate in *world coordinates*. World coordinates are a device-independent, 2 or 3D, Cartesian coordinate system for describing objects. Output primitives are given to *SunCore* functions in World Coordinates (WC). However, if the *world coordinate* matrix is used, *SunCore* concatenates this matrix with the view transform so that output primitives are first transformed by this matrix from 'model' or 'object' coordinates to world coordinates. This means that the user can supply primitives in 'model' coordinates, each model or object being moved into world coordinates according to the current *world coordinate matrix.*

In 3D, the user may choose to use right-handed or left-handed world coordinates. In a right-handed system, if (for example) the x coordinate increases to the right and the y coordinate increases upwards, then the z coordinate increases towards the viewer. In the corresponding left-handed system, the x coordinate increases to the right, the y coordinate increases upwards, and the z coordinate increases away from the viewer.

The composite viewing transform is formed from the *world coordinate matrix* and the viewing parameters. *SunCore* functions transform the output primitives from world (or model) coordinates to NDC, which is a left-hand coordinate system bounded such that: $0.0 \leq x, y, z \leq 1.0$

Since current Sun view surfaces have four-to-three aspect ratios, the default NDC space has the y extent bounded to $0.0 \leq y \leq 0.75$. Primitives are stored in the Display List (also called the Pseudo Display File or PDF), in NDC space. The user-specified window in world coordinates is mapped (and optionally clipped) to the user-specified viewport within NDC space. The entire NDC space is then mapped to the selected physical view surfaces.

1.6. Details of Using SunCore

This section describes the details of creating applications programs to run with *SunCore*.

Classification of Functional Capabilities

The ACM Core specification defines levels of functional capability for a graphics package which implements the specification. The table below shows the classification. Terms such as BUFFERED and DYNAMICA are defined as constants in the file `<usercore.h>`, discussed below.

Table 1-1 *Output Capabilities*

Functional Capability	*BASIC*	*BUFFERED*	*DYNAMICA*	*DYNAMICB*	*DYNAMICC*
Output Primitives and Primitive Attributes.	yes	yes	yes	yes	yes
Viewing	yes	yes	yes	yes	yes
Control	yes	yes	yes	yes	yes
Temporary Segments	yes	yes	yes	yes	yes
Retained Segments	no	yes	yes	yes	yes
Highlighting Segment Attribute	no	yes	yes	yes	yes
Visibility Segment Attribute	no	yes	yes	yes	yes
Image Transformation Segment Attribute	no	no	yes	yes	yes
Detectability Segment Attribute	no	yes*	yes*	yes*	yes*

* This feature is only available if input levels SYNCHRONOUS or COMPLETE are supported. Note that *SunCore* supports all output levels up to DYNAMICC.

Table 1-2 *Input Capabilities*

Functional Capability	*NOINPUT*	*SYNCHRONOUS*	*COMPLETE*
Device Initialization and Termination	no	yes	yes
Synchronous Interaction Functions	no	yes	yes
Echo Control	no	yes	yes
Explicit Enable or Disable	no	no	yes
Event Queue Management	no	no	yes
Sampled Device Functions	no	no	yes
Associations	no	no	yes

Note that *SunCore* supports up to the SYNCHRONOUS input level.

Table 1-3 *Dimension Levels Supported*

Functional Capability	*TWOD*	*THREED*
2D Primitives, Attributes, and Viewing.	yes	yes
3D Primitives, Attributes, and Viewing.	no	yes

Note that *SunCore* supports up to the THREED dimension level.

Error Reporting

SunCore performs consistency checks on arguments passed to its various functions. Any time an error is detected, the name of the function which raised the error condition and the text of the error message are printed on the standard error (stderr).

All *SunCore* interfaces are *functions* that return a value. If a function completes successfully, it returns the value zero. If the function raises any error conditions, it returns a non-zero value. *SunCore* always identifies the name of the function which raised the error condition. The ACM Core specification defines specific error numbers. These do not correspond to *SunCore*'s error numbers in the current release.

Useful Constants in the `<usercore.h>` Include File

The file `<usercore.h>` defines a collection of constants which the application programmer should use in lieu of hardwired constants in code. The constants are described here (but their values are not stated).

Useful Constants:

TRUE A universal value denoting the truth value.

FALSE A universal value denoting the false value.

MAXVSURF The maximum number of view surfaces which may be initialized at any one time.

Initialization Constants. These constants describe the levels of the *SunCore* facilities which the application program will use. These constants should be used when calling the `initialize_core()` function.

BASIC Denotes the basic output level. See the tables above for the classifications.

BUFFERED
Denotes the buffered output level. See the tables above for the classifications.

DYNAMICA
Indicates that the application package wishes to use 2D translation facilities. See the tables above for the classifications.

DYNAMICB
Indicates that the application package wishes to use 3D scaling, rotation, and translation facilities. See the tables above for the classifications.

DYNAMICC
: Indicates that the application package wishes to use 3D scaling, rotation, and translation facilities. See the tables above for the classifications.

NOINPUT
: Indicates that this application package will not use any input facilities. See the tables above for the classifications.

SYNCHRONOUS
: Indicates that this application program will use synchronous input facilities. See the tables above for the classifications.

COMPLETE
: *SunCore* does not support this input level. See the tables above for the classifications.

TWOD
: Indicates that the application package will only use 2D functions. See the tables above for the classifications.

THREED
: Indicates that the application package will use both 2D and 3D functions. See the tables above for the classifications.

Character Quality Constants. These constants should be used when calling the `set_charprecision()` function.

STRING
: Denotes low quality text.

CHARACTER
: Denotes medium quality text.

Transform Constants. These constants should be used when calling the `set_projection()` and `set_coordinate_system_type()` functions.

PARALLEL
: Value to indicate *parallel* projection.

PERSPECTIVE
: Value to indicate *perspective* projection.

RIGHT
: Value to indicate right-handed world coordinate system.

LEFT
: Value to indicate left-handed world coordinate system.

Image Transformation Type Constants. These constants are used when calling the `set_image_transformation_type()` and `set_segment_image_transformation_type()` functions.

NONE
: Indicates a retained segment which cannot be transformed.

XLATE2
: Indicates a retained segment which may be translated in 2D.

XFORM2
: Indicates a retained segment which may be fully translated, scaled, and rotated, in 2D.

XLATE3
: Indicates a retained segment which may be translated in 3D.

XFORM3
: Indicates a retained segment which may be fully translated, scaled, and rotated, in 3D.

Line Style Constants. These constants should be used when calling the `set_linestyle()` attribute for output primitives.

SOLID Solid line.

DOTTED Dotted line.

DASHED Dashed line.

DOTDASHED
Dashed and dotted line.

Text Font Selection Constants. These constants should be used when calling `set_font()`.

ROMAN For *character* precision, a Roman font; for *string* precision, a raster font.

GREEK For *character* precision, a Greek font; for *string* precision, the default raster font.

SCRIPT For *character* precision, a Script font; for *string* precision, a small raster font.

OLDENGLISH
For *character* precision, an Old English font; for *string* precision, equivalent to ROMAN.

STICK This is equivalent to a medium sized ROMAN raster font.

SYMBOLS This is equivalent to a bold version of STICK. It currently holds some electronics symbols (character values 32 through 47).

Input Device Constants. These constants should be used when calling the `initialize_device()` and `terminate_device()` functions and other input functions.

PICK The *Pick* device. The mouse in *SunCore*.

KEYBOARD
The *Keyboard* device.

STROKE The freehand STROKE device. The mouse in *SunCore*.

LOCATOR The *Locator* device. The mouse in *SunCore*.

VALUATOR
The *Valuator* device. The mouse in *SunCore*.

BUTTON The *Button* device. The mouse in *SunCore*.

RasterOp Constants. These constants should be used when calling the `set_rasterop()` function.

NORMAL Indicates normal copy mode.

XORROP Indicates bitwise exclusive or of source and destination.

ORROP Indicates bitwise or of source and destination.

Polygon Rendering Style Constants. These constants should be used when calling the `set_polygon_interior_style()` and `set_shading_parameters()` functions.

PLAIN Indicates area fill with the color indicated by the *fill index* primitive attribute.

SHADED Indicates shading according to the current shading parameters (for 3-D polygons only).

CONSTANT
Indicates constant user-specified shade.

GOURAUD
Indicates Gouraud shading.

PHONG Indicates Phong shading.

1.7. Further Reading

[1] Conrac Corporation. *Raster Graphics Handbook*, Second Edition. Van Nostrand Reinhold, 1985.

[2] J.D. Foley and A. Van Dam. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, 1982.

[3] W.M. Newman and R.F. Sproull. *Principles of Interactive Computer Graphics*. McGraw-Hill, 1979.

[4] ACM-SIGGRAPH. Conference Proceedings.

[5] IEEE Computer Graphics and Applications.

[6] *Status Report of the Graphics Standards Planning Committee.* Computer Graphics. Volume 13, Number 3, August 1979.

[7] *Special Issue on Graphics Standards.* ACM Computing Surveys. Volume 10, #4, December 1978.

[8] *The SIGGRAPH Core System Today.* Computer Graphics World. Volume 5, #8, August 1982.

[9] *SunView Programmer's Guide.* Sun Microsystems.

[10] *SunView System Programmer's Guide.* Sun Microsystems.

[11] *Pixrect Reference Manual.* Sun Microsystems.

[12] *SunCGI Reference Manual.* Sun Microsystems.

[13] *FORTRAN Programmer's Guide for the Sun Workstation.* Sun Microsystems.

[14] *Pascal Programmer's Guide for the Sun Workstation.* Sun Microsystems.

2

Control

2

Control

The *SunCore* graphics package provides several functions for controlling the system. These functions are discussed here, and the sections and subsections which follow describe the individual functions in detail.

Initialization and *termination*
of *SunCore* provide for the initialization of the package to a specific and predetermined state, and for closing it down when the applications program has finished using the graphics package.

View surface control
provides for the initialization, termination, and selection of view surfaces. A view surface must be initialized before it can be used. A view surface should be terminated when the applications package has finished with it. Functions are provided to add view surfaces to the set of selected view surfaces, and to remove view surfaces from that set. View surface names in *SunCore* are structures. The `vwsurf` structure is declared in `<usercore.h>` and is described in Appendix B. *SunCore* supports several view surfaces; see Appendix B for details of view surfaces.

Picture change control
provides for the "batching" of changes to dynamic segment attributes so that the application program may force the simultaneous occurrence of a group of changes.

Frame control
denotes the function called `new_frame()`, which clears the view surface and redraws all segments except temporary segments.

Error handling
is that part of *SunCore* concerned with reporting errors to the application program.

2.1. Initialization and Termination

There are two functions provided for initializing and terminating *SunCore*. The application program should call `initialize_core()` before making any other calls upon the graphics system. `terminate_core()` should be the last call to *SunCore* before the application program itself is finished.

Initialize the SunCore System

```
initialize_core(output_level, input_level, dimension)
int output_level;    /* SunCore Level for Output */
            /* BASIC, BUFFERED, DYNAMICA */
            /* DYNAMICB, DYNAMICC */
int input_level;     /* SunCore Level for Input */
            /* NOINPUT, SYNCHRONOUS, COMPLETE */
int dimension;       /* Number of Dimensions Required */
            /* TWOD, THREED */
```

`initialize_core()` initializes the Core graphics package to a known state.

SunCore supports up to output level DYNAMICC of the ACM Core specification, up to input level SYNCHRONOUS of the ACM Core, and dimension level THREED of the ACM Core.

- The *SunCore* system is already initialized.
- The specified output level cannot be supported.
- The specified input level cannot be supported.
- The specified dimension cannot be supported.

Close Down the SunCore System

```
terminate_core()
```

`terminate_core()` closes down the Core graphics package.

2.2. Initializing and Selecting View Surfaces

View surface control provides for the initialization, termination, and selection of view surfaces. A view surface must be initialized before it can be used. A view surface should be terminated when the applications package has finished with it. Examples of view surfaces are the Sun color display and the Sun monochrome bitmap display. Functions provided in this category are:

`initialize_view_surface`
: performs the functions required to gain access to a specified view surface.

`terminate_view_surface`
: terminates access to the specified view surface.

`select_view_surface`
: adds the specified view surface to the set of selected view surfaces for output.

`deselect_view_surface`
: removes the specified view surface from the set of selected view surfaces.

`inquire_selected_surfaces`
: determines which view surfaces are currently selected (not yet implemented).

Initialize a View Surface

```
initialize_view_surface(surface_name, type)
struct vwsurf *surface_name;     /* See Appendix B */
int type;    /* TRUE for hidden surface removal,
    FALSE otherwise */
```

`initialize_view_surface()` initializes the Core package for a specific view surface.

The `surface_name` argument to the function specifies a physical view surface. View surface names in *SunCore* are structures. The *vwsurf* structure is defined in the `<usercore.h>` header file. Only color or gray scale devices support hidden-surface removal.

- The view surface specified by `surface_name` is already initialized.
- The view surface specified by `surface_name` does not have any output device associated with it.
- No other view surfaces can be initialized at this time.
- The specified view surface does not support hidden surface removal.

Close Down a View Surface

```
terminate_view_surface(surface_name)
struct vwsurf *surface_name;     /* See Appendix B */
```

`terminate_view_surface()` closes down the specified view surface.

- The view surface specified by `surface_name` is not initialized.

Add View Surface to Selected Set

```
select_view_surface(surface_name)
struct vwsurf *surface_name;     /* See Appendix B */
```

`select_view_surface()` adds a specified view surface to the list of selected view surfaces.

A segment is only drawn on those view surfaces marked as "selected" at the time that the segment is created.

- A segment is open.
- The view surface specified by `surface_name` is not initialized.
- The view surface specified by `surface_name` is already selected.
- The view surface specified by `surface_name` cannot be selected.

Remove View Surface from Selected Set

```
deselect_view_surface(surface_name)
struct vwsurf *surface_name;     /* See Appendix B */
```

`deselect_view_surface()` removes a specified view surface from the list of selected view surfaces.

Segments created after `deselect_view_surface()` is called will not be drawn on the deselected view surface.

- A segment is open.
- The view surface specified by `surface_name` is not selected.

2.3. Batching of Updates

SunCore provides the facility for the application program to indicate that a sequence of updates is being started, and the graphics package stacks up these picture changes until an `end_batch_of_updates()` function call indicates that the end of the sequence of updates has occurred. Picture changes or 'updates' include dynamic segment attributes such as visibility, detectability, translate, rotate, and scale.

Indicate Start of a Batch of Updates

```
begin_batch_of_updates()
```

`begin_batch_of_updates()` indicates the beginning of a batch of updates to the picture. All modifications to dynamic attributes of segments between calls to `begin_batch_of_updates()` and `end_batch_of_updates()` are saved up and executed simultaneously.

- There has been no `end_batch_of_updates()` function call since the last `begin_batch_of_updates()` function call.

Indicate End of a Batch of Updates

```
end_batch_of_updates()
```

`end_batch_of_updates()` indicates the end of a batch of updates. The batch of changes to dynamic attributes of segments is executed,

- There has been no corresponding `begin_batch_of_updates()` function call.

Start New Frame Action for Selected View Surfaces

```
new_frame()
```

`new_frame()` starts new frame action for currently selected view surfaces. The view surface is cleared, and all visible retained segments are redrawn.

- The set of currently selected view surfaces is empty.

2.4. Error Control

The following functions control the display of error information. This information can be used to determine the source of an error.

Report Most Recent Error

```
report_most_recent_error(error_number)
int *error_number;
```

`report_most_recent_error()` obtains a copy of the most recently detected error number. A value of zero returned to `error_number` indicates that there has been no error since the last call on `report_most_recent_error()`.

Print Error

```
print_error("Your message", error_number);
int error_number;
```

`print_error()` prints the message associated with this `error_number` on the standard error file (stderr). *Your message* is any character string that the user wants printed. The error message is printed on the line following "Your message"

2.5. Miscellaneous

The following functions provide extensions to the Core System.

Drag Control (SunCore Extension)

```
set_drag(mode)
int mode;    /* FALSE = uses the rasterop */
             /* set by set_rasterop */
             /* TRUE = enable XOR'ing */
```

`set_drag()` writes all output to the bitmap or color framebuffer with XOR-ing. If dragging is enabled, all output to the device drivers is done with XOR's to the data in the displays. This feature makes dragging more convenient. For example, if you want to drag segment A across segment B, leaving segment B's image unaffected, do the following sequence of operations:

- Set A visibility *off*,
- Set dragging *on*,
- Set A visibility *on*,
- Drag segment A to the desired location,
- Set A visibility *off*,
- Set dragging *off*,
- Set A visibility *on*.

See also: `set_rasterop()`.

Signal Handling

SunCore uses the *SunView* Notifier to handle signals. Therefore, *SunCore* applications should use the Notifier instead of explicit `signal()` calls. See the manuals, *SunView Programmer's Manual* and *SunView System Programmer's Manual.*

3

Viewing Operations and Coordinate Transforms

3

Viewing Operations and Coordinate Transforms

Specifying a viewing operation may be thought of as specifying the arbitrary orientation of a synthetic camera. The resulting view of the object (the snapshot) can appear on one or more view surfaces. The viewing operations are provided for two reasons:

1. To specify how much of the world coordinate space should be visible, and
2. To specify a mathematical transformation between the world coordinate system and NDC space.

A viewing operation is specified by a view volume that defines the portion of world coordinate space which is to be projected onto a view plane (also called a projection plane), and a rectangular viewport in NDC space to which the projected image will be mapped. The viewing operation is sufficiently general as to support both parallel and perspective projections. The parallel projection includes the orthographic, axonometric, isometric, cavalier, and cabinet projections, as special cases.

Once the camera model is specified with `set_view_reference_point()`, `set_view_plane_normal()`, and so on, a 4×4 view transform matrix is constructed. Then the process of generating an image on a view surface is:

1. View-transforming the output primitives (using the view transform preceded by any modelling transform the user has specified) to NDC space.
2. Optional clipping to the window.
3. Scale the output to map the window to the viewport.
4. Optional image transformation as specified by dynamic segment attributes.
5. Optional clipping to the viewport.
6. Convert to device coordinates and draw the picture.

3.1. Windows, View Volumes, and Clipping

The *window* is the bounded portion of the view plane containing projected objects which will appear within the viewport on the *view surface*. The view surface corresponds to the physical device on which the picture is drawn. The window is the logical region, specified in world coordinates, in which the image appears.

Specifying a window involves defining a coordinate system for the *view plane*. The coordinate system for the view plane is called the *uvw* coordinate system, to distinguish it from the world coordinate system and the NDC space, both of which are *xyz* coordinate systems.

The origin of the *uvw* coordinate system is at the point where the line through the view reference point parallel to the view plane normal vector intersects the view plane. In the default case, the view plane distance is zero, and so the view reference point lies in the view plane and is the origin of the *uvw* coordinate system.

The direction of the *v*-axis is determined from the *view up vector*. The view up vector is specified in world coordinates relative to the view reference point.

The positive *u*-axis of the *uvw* coordinate system is 90 degrees clockwise from the positive *V* axis, as viewed in the direction of the view plane normal vector. The positive *U* and *V* axes, together with the view plane normal vector, form a left handed coordinate system. The window is specified in terms of maximum and minimum *u* and *v* values (see the `set_window()` function). Figure 3-1 shows the various components of the viewing system.

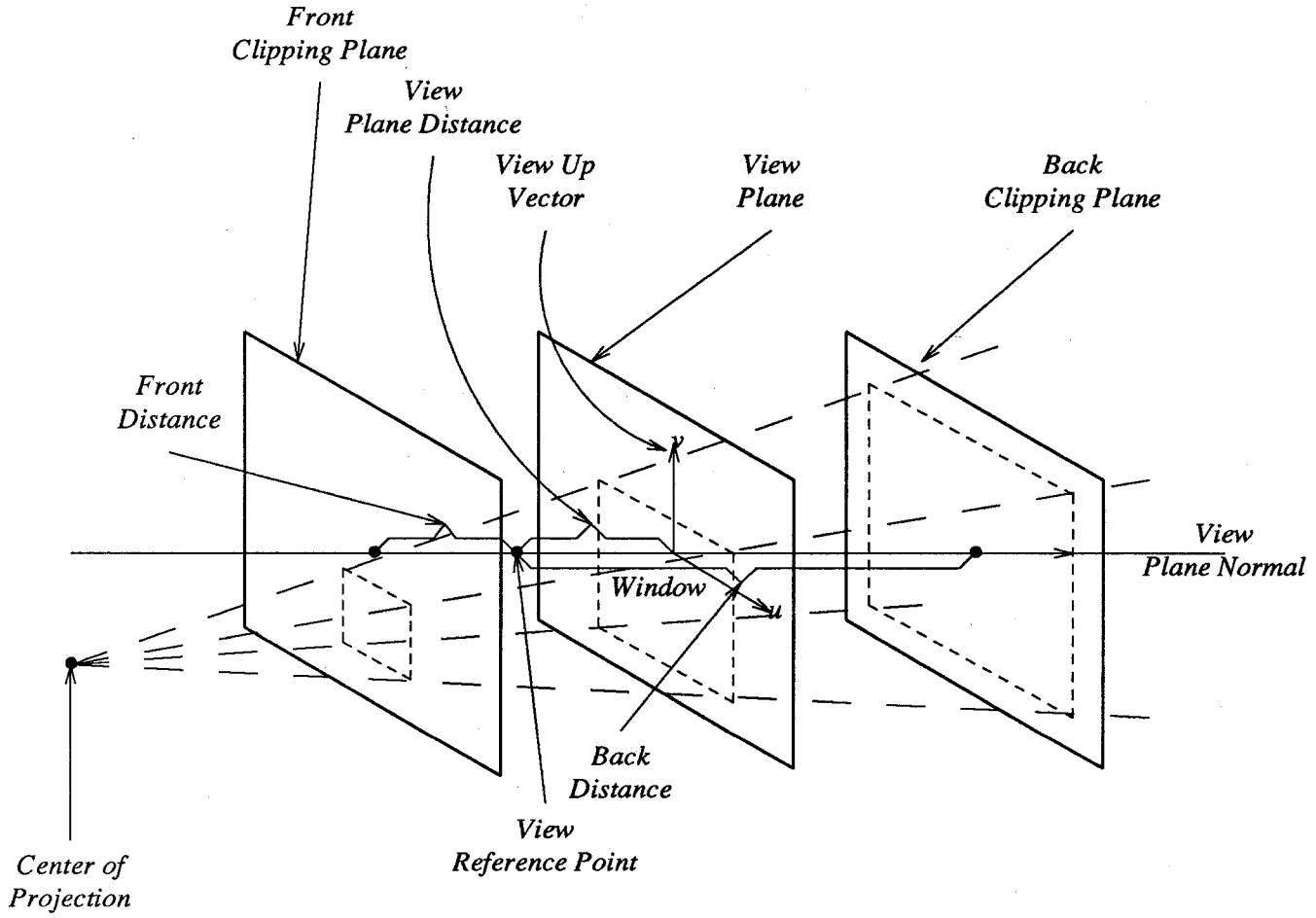


Figure 3-1 *Components of Viewing System*

3.2. Default Values of Viewing Operation Parameters

Table 3-1 *Default Values of Viewing Operation Parameters*

Parameter	*Default Value*
View Reference Point	{0, 0, 0}
View Plane Normal	{0, 0, −1}
View Distance	0
Front Distance	0
Back Distance	1
Type of Projection	*Parallel* (0, 0, 1) (perpendicular to the *uv* plane)
Window	(0, 1, 0, 0.75)
View Up Vector	(0, 1, 0)
NDC Space	$0.0 \le x, z \le 1.0$ $0.0 \le y \le 0.75$
Viewport	(0.0, 1.0, 0.0, 0.75, 0.0, 1.0)

Table 3-2 *Default Values of Viewing Control Parameters*

Parameter	*Default Value*
Window Clipping	On
Output Clipping	Off
Front Plane Clipping	Off
Back Plane Clipping	Off
World Coordinate System	Right handed

Table 3-3 *World Coordinate Matrix Parameters (Modelling Transform)*

Parameter	*Default Value*
Identity Matrix	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1

Table 3-4 *Image Transformation Parameters*

Parameter	*Default Value*
SX, SY, SZ	1, 1, 1 (no scaling)
AX, AY, AZ	0, 0, 0 (no rotation)
TX, TY, TZ	0, 0, 0 (no translation)

3.3. Setting 3D Viewing Operation Parameters

SunCore provides a number of functions for setting parameters of the viewing operations. There are a number of separate calls available for setting individual parameters, then there is a composite `set_viewing_parameters()` function which sets all the viewing parameters in one fell swoop. The individual calls provided are summarized here and described in detail in the subsections following.

Table 3-5 *Summary of Functions for Setting Viewing Control Parameters*

Function	*Description*
`set_view_reference_point`	Sets the view reference point in world coordinates.
`set_view_plane_normal`	Defines a vector which determines the view plane, relative to the view reference point.
`set_view_plane_distance`	Defines the view plane distance from the view reference point along the view plane normal vector.
`set_view_depth`	Defines the distance from the view reference point to the 'front' clipping plane (also known as the 'hither' or 'near' clipping plane) and the distance from the view reference point to the 'back' clipping plane (also known as the 'yon' or 'far' clipping plane).
`set_projection`	Selects perspective or parallel projection, and defines the center of projection (for PERSPECTIVE projection) or direction of projection (for PARALLEL projection).
`set_view_up_2` `set_view_up_3`	Establish the view up direction in the view plane for 2 or 3D viewing.
`set_window`	Establishes the window boundaries in the view plane.
`set_viewport_2` `set_viewport_3`	Establish the viewport boundaries in NDC space for 2 or 3D viewing.
`set_ndc_space_2` `set_ndc_space_3`	Establish the size of NDC space for 2 or 3D viewing.
`set_viewing_parameters`	is a composite function which does all of the above functions at one time.

None of the above calls have any effect until the next call upon the `create_retained_segment()` or `create_temporary_segment()` functions.

Establish Reference Point for Viewing

```
set_view_reference_point(x, y, z)
float x, y, z;  /* x, y, and z coordinates */
```

`set_view_reference_point()` sets the view reference point in world coordinates. *x*, *y*, and *z* are the coordinates of the view reference point. In the

absence of a specified reference point, the default view reference point is (0, 0, 0). The new reference point does not take effect until a new segment is created.

Establish View Plane Normal Vector

```
set_view_plane_normal(dx_norm, dy_norm, dz_norm)
float dx_norm, dy_norm, dz_norm;
```

`set_view_plane_normal()` defines a vector relative to the view reference point, in world coordinates. The view plane is perpendicular to the view plane normal vector. In the absence of any information to the contrary, *SunCore* establishes the view plane normal vector as (0, 0, −1). The new vector does not take effect until a new segment is created.

- No view plane normal direction can be established because *dx_norm*, *dy_norm*, and *dz_norm* are all zero.

Establish View Plane Distance

```
set_view_plane_distance(distance)
float distance;
```

`set_view_plane_distance()` establishes the view, or projection, plane. The view plane is perpendicular to the view plane normal vector, and is *distance* from the view reference point along the view plane normal vector. Distances are measured in world coordinate units from the view reference point. Positive values of *distance* correspond to the direction of the view plane normal vector, and negative values correspond to the opposite direction. In the absence of any information to the contrary, *distance* is set to zero, which means that the viewing plane is located at the view reference point.

Select Projection Type

```
set_projection(projection, dx_proj, dy_proj, dz_proj)
int projection; /* Projection type */
        /* PARALLEL; PERSPECTIVE */
float dx_proj, dy_proj, dz_proj;
        /* x, y, and z Deltas of Projection Point */
```

`set_projection()` selects the projection system for displaying. The arguments *dx_proj*, *dy_proj*, and *dz_proj* specify a world coordinate point relative to the view reference point. If projection is PARALLEL, objects project onto the view plane along lines parallel to the vector specified by *dx_proj, dy_proj , and dz_proj. If projection is PERSPECTIVE, (dx_proj, dy_proj, dz_proj) specify a point in world coordinates relative to the reference point called the center of projection (often abbreviated to COP). Objects project onto the view plane along lines travelling towards this point. Thus the center of projection is the apex of a pyramid whose edges pass through the four corners of the view window.*

- The direction of projection cannot be established because *dx*, *dy*, and *dz* are all zero. Note that this error is only applicable if parallel projection was selected.

Establish 2D View Up Vector

```
set_view_up_2(dx, dy)
float dx, dy;   /* dx and dy coordinates */
```

`set_view_up_2()` establishes a view up vector in 2D. This vector defines the direction of 'up' for the window in world coordinates.

- The view up vector cannot be established because *dx*, and *dy* are both zero.

Establish 3D View Up Vector

```
set_view_up_3(dx_up, dy_up, dz_up)
float dx_up, dy_up, dz_up;
    /* x, y, and z Deltas of View Up Vector */
```

`set_view_up_3()` establishes a view up vector in 3D. The three arguments *dx_up*, *dy_up*, and *dz_up* establish a view up vector relative to the view reference point. The view up vector, when projected onto the view plane in the direction of the view plane normal vector, specifies the positive *v*-axis of the *uvw* coordinate system in the view plane. The *u*-axis is also in the view plane, such that the *u*-axis, the *v*-axis, and the view plane normal vector form a left handed coordinate system. The *v*-axis is vertical and the *u*-axis increases to the right when the view plane is mapped onto the view surface.

SunCore establishes the default view up vector as (0, 1, 0), which means that the *y*-axis is *up*.

If the view plane normal vector is parallel to the *y*-axis, this does not work and so *SunCore* checks the view transforms for validity when creating a segment. *SunCore* may generate the error message:

```
'The current viewing specification is inconsistent'
```

- No view plane normal direction can be established because *dx_up*, *dy_up*, and *dz_up* are all zero.

Establish Size of 2D NDC Space

```
set_ndc_space_2(width, height)
float width, height;
```

`set_ndc_space_2()` defines the size of the NDC space which can be addressed on the view surface of all display devices available to the applications program and within which viewports may be established. Both *width* and *height* must be in the range of 0.0 to 1.0, and at least one of the parameters must have a value of 1.0. NDC space ranges from 0.0 to *width* in the horizontal direction and from 0.0 to *height* in the vertical direction. The rectangle defined by this function is mapped to the viewable area of any display device available to the application program so that the entire rectangle is visible. Only uniform scaling of the rectangle is allowed; no changes can be made to the viewport aspect ratio. *SunCore* maximizes the usable area of the display and centers NDC space on each view surface.

The default NDC specification is *width*=1.0 and *height*=0.75. Either of the `set_ndc_space_2()` or `set_ndc_space_3()` (see below) functions may be used at most once per initialization of *SunCore*, and the NDC space established applies to all view surfaces which the application program might use.

Ten *SunCore* functions require that NDC space be established before they complete execution. If NDC space has not been explicitly defined before any of these functions are executed, they implicitly define the NDC space using default values. Functions which implicitly define NDC space are:

- `initialize_device()`
- `initialize_group()`
- `create_retained_segment()`
- `create_temporary_segment()`
- `set_viewport_2()`
- `set_viewport_3()`
- `set_viewing_parameters()`
- `inquire_viewport_2()`
- `inquire_viewport_3()`
- `inquire_viewing_parameters()`

The *depth* of NDC space is set to 0.0 if `set_ndc_space_2()` is used in a 3D implementation.

- `set_ndc_space_2()` or `set_ndc_space_3()` has already been called since the system was initialized.
- `set_ndc_space_2()` or `set_ndc_space_3()` has been called too late — the default values have already been defined implicitly.
- A parameter is outside the range 0.0 to 1.0.
- One of *width* or *height* must have a value of 1.0.
- *width* or *height* has a value of 0.0.

Establish Size of 3D NDC Space

```
set_ndc_space_3(width, height, depth)
float width, height, depth;
```

`set_ndc_space_3()` defines the size of the NDC space which can be addressed on the view surface of all display devices available to the applications program and within which viewports may be established. 3D NDC space is a rectangular parallelepiped lying within the NDC system. This coordinate system is always left-handed, with the *x*-axis increasing to the right, the *y*-axis increasing upwards, and the *z*-axis increasing away from the viewer. All of the parameters *width*, *height*, and *depth* must be in the range of 0.0 to 1.0, and at least one of *width* or *height* must have a value of 1.0. NDC space ranges from 0.0 to *width* in the horizontal direction, from 0.0 to *height* in the vertical direction, and from 0.0 to *depth* in the direction away from the viewer. The rectangle of size *width* by *height* in the *z*=0 plane of NDC space is mapped to the viewable area of any display device available to the application program so that the entire rectangle is visible. Only uniform scaling of the rectangle is allowed — no changes can be made to the viewport aspect ratio. *SunCore* maximizes the usable area of the display and centers NDC space on each view surface.

The default NDC specification is *width*=1.0, *height*=0.75, and *depth*=1.0. Either of the `set_ndc_space_3()` or `set_ndc_space_2()` (see above) functions may be used at most once per initialization of *SunCore*, and the NDC space established applies to all view surfaces which the application program might use.

Ten *SunCore* functions require that NDC space be established before they complete execution. If NDC space has not been explicitly defined before any of these functions are executed, they implicitly define the NDC space using default values. Functions which implicitly define NDC space are:

- `initialize_device()`
- `initialize_group()`
- `create_retained_segment()`
- `create_temporary_segment()`
- `set_viewport_2()`
- `set_viewport_3()`
- `set_viewing_parameters()`
- `inquire_viewport_2()`
- `inquire_viewport_3()`
- `inquire_viewing_parameters()`.

- `set_ndc_space_2()` or `set_ndc_space_3()` has already been called since the system was initialized.
- `set_ndc_space_2()` or `set_ndc_space_3()` has been called too late — the default values have already been defined implicitly.
- A parameter is outside the range 0.0 to 1.0.
- One of *width* or *height* must have a value of 1.0.
- *width* or *height* has a value of 0.0.

Establish a Window in the View Plane

```
set_window(umin, umax, vmin, vmax)
float umin, umax;    /* Left and Right sides of window */
float vmin, vmax;    /* Bottom and Top of window */
```

`set_window()` establishes a window, defined by four coordinates in the *uv* coordinate system, in the view plane. *SunCore* establishes the default window as (0.0, 1.0, 0.0, 0.75).

- *umin* is greater than or equal to *umax*, which means that the left side of the window is congruent with or to the right of the right side of the window.
- *vmin* is greater than or equal to *vmax*, which means that the top of the window is congruent with or below the bottom of the window.

Specify Planes for Depth Clipping

```
set_view_depth(front_distance, back_distance)
float front_distance, back_distance;
    /* Distances to Front and Back Planes */
```

`set_view_depth()` defines the front and back planes for depth clipping. Clipping to these depth bounds is controlled by `set_front_plane_clipping()` and `set_back_plane_clipping()`. The front and back planes determine the

3D view volume which is mapped to the 3D viewport.

SunCore initializes the front distance to 0.0 and the back distance to 1.0.

- *front_distance* is greater than *back_distance*, so that the back clipping plane is in front of the front clipping plane.

Establish Limits of 2D Viewport

```
set_viewport_2(xmin, xmax, ymin, ymax)
float xmin, xmax; /* Left and Right sides of Viewport */
float ymin, ymax; /* Bottom and Top of Viewport */
```

`set_viewport_2()` establishes the limits of the viewport in 2D NDC space. The limits must lie in the range: $0 \leq x \leq NDCwidth$ and $0 \leq y < .SMNDCheight$ *SunCore* establishes the viewport to (0.0, 1.0, 0.0, 0.75) at initialization time.

- *xmin* is greater than or equal to *xmax*, which means that the left side of the viewport is congruent with or to the right of the right side of the viewport.
- *ymin* is greater than or equal to *ymax*, which means that the top of the viewport is congruent with or below the bottom of the viewport.
- Viewport exceeds NDC space.

Establish Limits of 3D Viewport

```
set_viewport_3(xmin, xmax, ymin, ymax, zmin, zmax)
float xmin, xmax; /* Left and Right sides of Viewport */
float ymin, ymax; /* Bottom and Top of Viewport */
float zmin, zmax; /* Front and Back of Viewport */
```

`set_viewport_3()` establishes the limits of the viewport in 3D NDC space. The limits must lie in the range: $0 \leq x \leq NDCwidth$, $0 \leq y < .SMNDCheight$, and $0 \leq z < NDCdepth$ *SunCore* establishes the viewport to (0.0, 1.0, 0.0, 0.75, 0.0, 1.0) at initialization time.

- *xmin* is greater than or equal to *xmax*, which means that the left side of the viewport is congruent with or to the right of the right side of the viewport.
- *ymin* is greater than or equal to *ymax*, which means that the top of the viewport is congruent with or below the bottom of the viewport.
- *zmin* is greater than or equal to *zmax*, which means that the front of the viewport is congruent with or behind the back of the viewport.
- Viewport exceeds NDC space.

Set Viewing Parameters

```
set_viewing_parameters(view_parameters)
struct {
    float vwrefpt[3]; /* x, y, z */
    float vwplnorm[3]; /* dx, dy, dz */
    float viewdis; /* View Reference Point to View Plane */
    float frontdis; /* View Reference Point to Front Clip Plane
    float backdis; /* View Reference Point to Back Clip Plane *
    int projtype; /* PARALLEL or PERSPECTIVE */
    float projdir[3]; /* Meaning depends on projection type */
    float window[4]; /* umin, umax, vmin, vmax */
    float vwupdir[3]; /* dx, dy, dz */
    float viewport[6]; /* xmin, xmax, ymin, ymax, zmin, zmax */
} *view_parameters;
```

`set_viewing_parameters()` specifies all the viewing parameters with a single function call. The *view_parameters* argument is a pointer to a structure as defined above. `set_viewing_parameters()` fills in the associated structure with the current values of the viewing parameters. The parameters are:

vwrefpt An array of three `floats` describing the coordinates of the view reference point.

vwplnorm An array of three `floats` describing the direction of the view plane normal vector.

viewdis A float describing the distance of the view plane from the view reference point.

frontdis A float describing the front clipping distance.

backdis A float describing the back clipping distance.

projtype A `int` describing the projection type.

projdir An array of three `floats` describing the direction of projection. The meaning of *projdir* is dependent on the projection type:

PARALLEL *projdir* specifies the direction of projection.

PERSPECTIVE *projdir* specifies the center of projection.

window An array of four `floats` describing the boundaries of the viewing window.

vwupdir An array of three `floats` describing the view up direction.

viewport An array of six `floats` describing the boundaries of the viewport.

3.4. Viewing Control

The functions described in the following sections allow the user to control viewing attributes like clipping and coordinate systems.

Enable Clipping in the View Plane

```
set_window_clipping(on_off)
int on_off; /* TRUE = turn clipping on */
            /* FALSE = turn clipping off */
```

`set_window_clipping()` enables or disables clipping against the window in the view plane. The *on_off* argument specifies whether window clipping is enabled or not. A value of FALSE *disables* window clipping, whereas a value of TRUE *enables* window clipping.

When window clipping is *off*, objects described to *SunCore* are not checked to insure that they lie within the window when projected onto the view plane. When window clipping is *on*, objects described to *SunCore* are clipped to the window.

SunCore initializes window clipping to TRUE.

Note that window clipping is done before segment primitives are written to the pseudo display file. This means that subsequent image transformations may extend images beyond the bounds of the viewport. *SunCore* has optional output clipping (an extension to the ACM Core specification) to correct for this. See the `set_output_clipping()` function described below.

Enable Front Plane Depth Clipping

```
set_front_plane_clipping(front_on_off)
int front_on_off;
```

`set_front_plane_clipping()` enables or disables clipping against the front clipping plane. The *front_on_off* argument specifies clipping enabled or disabled for the front clipping plane. A value of FALSE means *disable* the clipping, and a value of TRUE *enables* the clipping. Clipping is disabled by default.

Enable Back Plane Depth Clipping

```
set_back_plane_clipping(back_on_off)
int back_on_off;
```

`set_back_plane_clipping()` enables or disables clipping against the back clipping plane. The *back_on_off* argument specifies clipping enabled or disabled for the back clipping plane. A value of FALSE means *disable* the clipping, and a value of TRUE *enables* the clipping. Clipping is disabled by default.

Set Output Clipping (SunCore extension)

```
set_output_clipping(on_off)
int on_off; /* TRUE = turn on clipping */
            /* FALSE = turn off clipping */
```

SunCore supports output clipping, which is done after image transformations on segments, as an option in addition to window clipping. The `set_output_clipping()` function enables or disables output clipping. If output clipping is enabled, it places a clipping process after the image transformation specified by the dynamic segment attribute. This ensures that everything is correctly clipped to the viewport.

Set Coordinate System Type

```
set_coordinate_system_type(type)
int type;   /* RIGHT = right handed coordinates */
        /* LEFT = left handed coordinates */
```

`set_coordinate_system_type()` selects a left-handed or right-handed world coordinate system.

Specify 2D World or Modelling Transform

```
set_world_coordinate_matrix_2(array)
float array[3][3];  /* [row] [column] */
```

`set_world_coordinate_matrix_2()` specifies a 3×3 matrix containing the 'world transform' or modelling transform. This matrix is concatenated with the 'viewing transform' to give the 'composite viewing transform'. The composite viewing transform is the transform that is actually used for all *SunCore* viewing transform operations. The default world coordinate matrix is the identity matrix. Currently, this function does not modify column 2 of the matrix. This function may be called at any time, even in the midst of putting output primitives into a segment.

Note that the matrix order is such that:

$$xnew = x*array_{0,0} + y*array_{1,0} + array_{2,0}$$

$$ynew = x*array_{0,1} + y*array_{1,1} + array_{2,1}$$

Specify 3D World or Modelling Transform

```
set_world_coordinate_matrix_3(array)
float array[4][4];  /* [row] [column] */
```

`set_world_coordinate_matrix_3()` specifies a 4×4 matrix containing the 'world transform' or modelling transform. This matrix is concatenated with the 'viewing transform' to give the 'composite viewing transform'. The composite viewing transform is the transform that is actually used for all *SunCore* viewing transform operations. The default world coordinate matrix is the identity matrix. Currently, this function does not modify column 3 of the matrix. This function may be called at any time, even in the midst of putting output primitives into a segment.

Note that the matrix order is such that:

$$xnew = x*array_{0,0} + y*array_{1,0} + z*array_{2,0} + array_{3,0}$$

$$ynew = x*array_{0,1} + y*array_{1,1} + z*array_{2,1} + array_{3,1}$$

$$znew = x*array_{0,2} + y*array_{1,2} + z*array_{2,2} + array_{3,2}$$

Convert 2D NDC to World Coordinates

```
map_ndc_to_world_2(ndcx, ndcy, wldx, wldy)
float ndcx, ndcy;
float *wldx, *wldy;
```

`map_ndc_to_world_2()` maps a point in NDC space to its world coordinates.

Convert 3D NDC to World Coordinates

```
map_ndc_to_world_3(ndcx, ndcy, ndcz, wldx, wldy, wldz)
float ndcx, ndcy, ndcz;
float *wldx, *wldy, *wldz;
```

`map_ndc_to_world_3()` maps a point in NDC space to its world coordinates.

Convert 2D World to NDC Coordinates

```
map_world_to_ndc_2(wldx, wldy, ndcx, ndcy)
float wldx, wldy;
float *ndcx, *ndcy;
```

`map_world_to_ndc_2()` maps a point in world coordinates to its NDC space.

Convert 3D World to NDC Coordinates

```
map_world_to_ndc_3(wldx, wldy, wldz, ndcx, ndcy, ndcz)
float wldx, wldy, wldz;
float *ndcx, *ndcy, *ndcz;
```

`map_world_to_ndc_3()` maps a point in world coordinates to its NDC space.

3.5. Inquiring Viewing Characteristics

SunCore provides a number of functions for inquiring about parameters of the viewing operations. There are a number of separate calls available for inquiring about individual parameters, then there is a composite `inquire_viewing_parameters()` function which obtains all the viewing parameters in one fell swoop. The individual calls provided are summarized here and described in detail in the subsections following.

Table 3-6 *Summary of Functions for Inquiring Viewing Parameters*

Function	*Description*
`inquire_view_reference_point`	Obtains the view reference point in world coordinates.
`inquire_view_plane_normal`	Obtains a vector which determines the view plane, relative to the view reference point.
`inquire_view_plane_distance`	Obtains the distance from the view reference point to the view plane.
`inquire_view_depth`	Obtains the distance from the view reference point to the 'front' clipping plane (also known as the 'hither' or 'near' clipping plane), and the distance from the view reference point to the 'back' clipping plane (also known as the 'yon' or 'far' clipping plane).
`inquire_projection`	Determines which projection type is in use, and returns either the center of projection (for PERSPECTIVE projection) or direction of projection (for PARALLEL projection).
`inquire_view_up_2`	Determines the view up direction in 2D.
`inquire_view_up_3`	Determines the view up direction in 3D.
`inquire_viewport_2`	Obtains the coordinates of the 2D viewport.
`inquire_viewport_3`	Obtains the coordinates of the 3D viewport.
`inquire_window`	Obtain the boundaries of the viewing window.
`inquire_viewing_parameters`	is a composite function which does all of the above functions at one time.
`inquire_ndc_space_2`	Determine the size of the NDC space in 2D.
`inquire_ndc_space_3`	Determine the size of the NDC space in 3D.

Inquire View Reference Point

```
inquire_view_reference_point(x, y, z)
float *x, *y, *z;   /* x, y, and z Coordinates */
```

`inquire_view_reference_point()` obtains the coordinates of the view reference point.

Inquire View Plane Normal

```
inquire_view_plane_normal(dx, dy, dz)
float *dx, *dy, *dz;    /* x, y, and z deltas */
```

`inquire_view_plane_normal()` obtains the coordinates of the view plane normal vector.

Inquire View Plane Distance

```
inquire_view_plane_distance(view_distance)
float *view_distance;
```

`inquire_view_plane_distance()` obtains the distance of the view

plane from the view reference point.

Inquire View Depth

```
inquire_view_depth(front_distance, back_distance)
float *front_distance, *back_distance;
```

`inquire_view_depth()` obtains the distances of the front and back clipping planes from the view reference point.

Inquire Projection

```
inquire_projection(projection_type, dx, dy, dz)
int *projection_type;
float *dx, *dy, *dz;    /* x, y, and z deltas */
```

`inquire_projection()` obtains the current projection type and the coordinates of the center of projection (for PERSPECTIVE projections) or the direction of projection (for PARALLEL projections).

Inquire View Up 2

```
inquire_view_up_2(dx, dy)
float *dx, *dy; /* x and y directions */
```

`inquire_view_up_2()` obtains the view up direction in 2D.

Inquire View Up 3

```
inquire_view_up_3(dx, dy, dz)
float *dx, *dy, *dz;    /* x, y, and z directions */
```

`inquire_view_up_3()` obtains the view up direction in 3D.

Inquire NDC Space 2

```
inquire_ndc_space_2(width, height)
float *width, *height;
```

`inquire_ndc_space_2()` obtains the dimensions of the 2D NDC space.

Inquire NDC Space 3

```
inquire_ndc_space_3(width, height, depth)
float *width, *height, *depth;
```

`inquire_ndc_space_3()` obtains the dimensions of the 3D NDC space.

Inquire Viewport 2

```
inquire_viewport_2(xmin, xmax, ymin, ymax)
float *xmin, *xmax;
float *ymin, *ymax;
```

`inquire_viewport_2()` obtains the coordinates of the 2D viewport.

Inquire Viewport 3

```
inquire_viewport_3(xmin, xmax, ymin, ymax, zmin, zmax)
float *xmin, *xmax;
float *ymin, *ymax;
float *zmin, *zmax;
```

`inquire_viewport_3()` obtains the coordinates of the 3D viewport.

Inquire Window

```
inquire_window(umin, umax, vmin, vmax)
float *umin, *umax;
float *vmin, *vmax;
```

`inquire_window()` obtains the boundaries of the viewing window.

Inquire Viewing Parameters

```
inquire_viewing_parameters(view_parameters)
struct {
    float vwrefpt[3]; /* x, y, z */
    float vwplnorm[3]; /* dx, dy, dz */
    float viewdis; /* View Reference Point to View Plane */
    float frontdis; /* View Reference Point
        to Front Clip Plane */
    float backdis; /* View Reference Point
        to Back Clip Plane */
    int projtype; /* PARALLEL or PERSPECTIVE */
    float projdir[3]; /* Meaning depends
        on projection type */
    float window[4]; /* umin, umax, vmin, vmax */
    float vwupdir[3]; /* dx, dy, dz */
    float viewport[6]; /* xmin, xmax, ymin,
        ymax, zmin, zmax */
} *view_parameters;
```

`inquire_viewing_parameters()` returns a collection of information pertaining to the current parameters of the viewing system. The *view_parameters* argument is a pointer to a structure as defined above. `inquire_viewing_parameters()` fills in the associated structure with the current values of the viewing parameters. The parameters are:

vwrefpt An array of three `floats` describing the coordinates of the view reference point.

vwplnorm An array of three `floats` describing the direction of the view plane normal vector.

viewdis A `float` describing the distance of the view plane from the view reference point.

frontdis A `float` describing the front clipping distance.

backdis A `float` describing the back clipping distance.

projtype A `int` describing the projection type.

projdir An array of three `floats` describing the direction of projection. The meaning of *projdir* is dependent on the projection type:

PARALLEL
projdir specifies the direction of projection.

PERSPECTIVE
projdir specifies the center of projection.

window An array of four `floats` describing the boundaries of the viewing window.

vwupdir An array of three `floats` describing the view up direction.

viewport An array of six `floats` describing the boundaries of the viewport.

Inquire World Coordinate Matrix 2

```
inquire_world_coordinate_matrix_2(array)
float array[3][3];  /* array[row][col] */
```

`inquire_world_coordinate_matrix_2()` returns a 3 by 3 matrix containing the 'world transform' or modelling transform. This matrix is concatenated with the 'viewing transform' to give the 'composite viewing transform'. The composite viewing transform is the transform that is actually used for all *SunCore* viewing transform operations. The default world coordinate matrix is the identity matrix.

Inquire World Coordinate Matrix 3

```
inquire_world_coordinate_matrix_3(array)
float array[4][4];  /* array[row][col] */
```

`inquire_world_coordinate_matrix_3()` returns a 4 by 4 matrix containing the 'world transform' or modelling transform. This matrix is concatenated with the 'viewing transform' to give the 'composite viewing transform'. The composite viewing transform is the transform that is actually used for all *SunCore* viewing transform operations. The default world coordinate matrix is the identity matrix.

Inquire Inverse Composite Matrix (SunCore Extension)

```
inquire_inverse_composite_matrix(array)
float array[4][4];  /* array[row][col] */
```

SunCore uses the matrix inverse of the composite viewing transform internally for operations such as `map_ndc_to_world()`. This matrix may at times be useful to the applications program.

Inquire Viewing Control Parameters

```
inquire_viewing_control_parameters(windowclip,
    frontclip, backclip, type)
int *windowclip; /* TRUE if window clipping enabled */
int *frontclip; /* TRUE if front plane clipping enabled */
int *backclip; /* TRUE if back plane clipping enabled */
int *type; /* RIGHT or LEFT world coordinate system type */
```

`inquire_viewing_control_parameters()` obtains the enabled status of clipping, and the type of world coordinates in use.

4

Segmentation and Naming

4

Segmentation and Naming

All output primitives for a graphical object are placed in a *segment* by *SunCore* on request from the application program. Each segment defines an *image* which is a view of the object and which is part of the picture displayed on the view surface. An application program describes an object by creating a segment, calling output primitive functions (the results of which are placed in the segment), and then closing the segment.

There are two kinds of segments, namely: *temporary* segments and *retained* segments. Retained segments have an *image transformation type* which specifies how they can be transformed. Retained segments can be made visible or invisible, detectable (via the *pick* input function) or undetectable, highlighted, and may be transformed, depending on their type.

Retained segments have names (actually numeric identifiers) so that by placing output primitives in such segments, the application programmer can selectively modify parts of the picture by deleting and recreating segments (which effectively replaces them) so that their images change. Retained segments are stored in the display list for later dynamic modification.

Temporary segments are not saved in the display list, are only drawn once, and may not be modified dynamically. A *new frame* action deletes all portions of any temporary segments which have already been drawn.

4.1. Retained Segment Attributes

In the same way that primitive attributes affect the output primitives, *retained segment dynamic attributes* affect the characteristics of retained segments. From now on, the term *dynamic attributes* means the dynamic attributes of retained segments.

As well as being identified by the name of the retained segment into which they have been placed, output primitives may also be assigned a primitive attribute known as a *pick identifier* or *pick-id*. This means that within the single level of segmentation, another level of naming is provided. An example of the use of pick-id might be that all the character strings for (say) a menu could appear in a single segment, where each character string is assigned a different pick-id. Then when the user is using the mouse to select a specific item from the menu, the application program uses the PICK input function to find out which menu item was selected.

Retained segments have one *static* attribute and four *dynamic* attributes. Attributes, and the means of setting them and inquiring their values, are described in detail in Chapter 6.

The only *static* attribute of retained segments is the *image transformation type*. This attribute can have one of five values:

None
The segment is a retained segment on which no transformations may be applied.

Translatable 2D
The segment is a retained segment which may be translated in 2D.

Transformable 2D
The segment is a retained segment which may be fully translated, scaled, and rotated, in 2D.

Translatable 3D
The segment is a retained segment which may be translated in 2 or 3D.

Transformable 3D
The segment is a retained segment which may be fully translated, scaled, and rotated, in 2 or 3D.

SunCore sets *image transformation type* to the default value of NONE at initialization time.

The four *dynamic* attributes of retained segments are defined here.

Visibility
indicates whether the segment should have a visible image. There are only two values of this attribute, namely: TRUE and FALSE.

SunCore sets the default value of *visibility* to TRUE at initialization time.

Highlighting
indicates whether the segment's image should be highlighted. In *SunCore*, highlighting is done by blinking. There are only two values of the *highlighting* attribute, namely: TRUE and FALSE. When highlighting is turned on, the segment is blinked once.

SunCore sets the default value of *highlighting* to FALSE at initialization time.

Detectability
indicates whether the retained segment can be detected by the pick device (mouse pointing device). See the `await_pick()` function. The values for the *detectability* attribute, are: 0 through 2,147,483,647. *SunCore* sets the default value of *detectability* to 0 at initialization time.

Image Transformation
indicates how the image of a retained segment, in NDC space, is scaled, rotated, or translated. A segment's static *image transformation type* attribute limits the values which its *image transformation* attribute may have. See the set of functions called `set_segment_image_`*xxx*() in Chapter 6.

SunCore sets the default value of *image transformation* to the identity transformation at initialization time.

4.2. Retained Segment Operations

A retained segment is a form of storage for graphical primitives. This kind of segment remains for the duration of a *SunCore* application program unless it is deleted. After the program exits the contents of a retained segment are lost.

Create a New Segment

```
create_retained_segment(segment_name)
int segment_name;    /* Segment Identifier */
```

`create_retained_segment()` creates a new, empty, open segment. The `segment_name` argument defines a segment number in the range 1 through 2,147,483,647.

The image transformation type for the newly created segment is obtained from the current attribute value for `image_transformation_type`. The dynamic attribute values for the newly created segment are obtained from the default values of the dynamic attributes for retained segments.

Use the `set_image_transformation_type()` function, before calling `create_retained_segment()`, to specify whether the created segment is translatable or transformable. After calling `create_retained_segment()`, the specified segment is said to be "open". This means that output primitives can now be called upon to add graphics primitives (lines, text, polygons, and so on) to this segment.

Only one segment can be open at a time.

- The set of currently selected view surfaces is empty.
- The current viewing specification is inconsistent.
- There is already an open segment.
- A retained segment named `segment_name` already exists.
- The default value of `image_transformation` is invalid for the current `image_transformation_type`.

Close a Segment

```
close_retained_segment()
```

`close_retained_segment()` closes the currently open segment. Dynamic segment attributes may be changed both before and after closing the segment.

- There is no open retained segment.

Delete a Retained Segment

```
delete_retained_segment(segment_name)
int segment_name;    /* Segment Identifier */
```

`delete_retained_segment()` deletes a specifically named segment. The segment specified by the `segment_name` argument is deleted. If the segment being deleted is the currently open segment, it is closed before it is deleted. The deleted segment is erased from all view surfaces.

- There is no retained segment with the name `segment_name`.

Rename a Retained Segment

```
rename_retained_segment(segment_name, newname)
int segment_name;    /* Old Segment Identifier */
int newname;         /* New Segment Identifier */
```

`rename_retained_segment()` changes the name of a retained segment. The segment whose identity is `segment_name` is renamed as *newname*, and this name must be used in any future references to that segment. The segment `segment_name` is no longer accessible.

- There is no retained segment with the name `segment_name`.
- There is an existing retained segment named `new_name`.

Delete All Retained Segments

```
delete_all_retained_segments()
```

`delete_all_retained_segments()` deletes all retained segments. All retained segments are deleted. If there is a currently open retained segment, it is closed before it is deleted.

Inquire Retained Segment Surfaces

```
inquire_retained_segment_surfaces(segment_name,
    array_size, view_surface_array, number_of_surfaces)
int segment_name;    /* Name of Segment */
int array_size;      /* Size of View Surface Array */
struct vwsurf view_surface_array[];
    /* Array of view surface names */
int *number_of_surfaces;
    /* Returned number of surfaces */
```

`inquire_retained_segment_surfaces()` obtains the number and names of the view surfaces upon which this segment gets drawn. These view surfaces were 'selected' when the segment was created. The number of view surfaces selected at the time the retained segment name given by `segment_name` was created is copied into `number_of_surfaces`. The names of those surfaces are copied into `view_surface_array`, where the array is an array of view surface names. `array_size` is specified by the caller, and is the size of `view_surface_array`. The view surface structure is defined in the `<usercore.h>` header file.

If `number_of_surfaces` is greater than *array_size*, only `array_size` view surface names are copied into *view_surface_array*. If `array_size` is less than or equal to zero, no names are returned.

- There is no retained segment with the name `segment_name`.

Inquire Retained Segment Names

```
inquire_retained_segment_names(array_size,
    name_array, number_of_segments)
int array_size;      /* Size of Array */
int name_array[];    /* Segment Identifiers */
int *number_of_segments;     /* Number of Segments */
```

`inquire_retained_segment_names()` obtains a list of the retained segments names. The `name_array` argument is an array which is to receive a list

of the existing retained segments. `array_size` specifies the number of elements in `name_array`. The *number_of_segments* argument is returned to the caller, and is the number of existing retained segments. If the number of existing retained segments is greater than the size of the array, only `array_size` segment names are copied into the array. If `array_size` is less than or equal to zero, no segment identifiers are returned.

Inquire Open Retained Segment

```
inquire_open_retained_segment(segment_name)
int *segment_name;  /* Segment Name */
```

`inquire_open_retained_segment()` obtains the name of the currently open retained segment. The name of the currently open retained segment (if there is one) is copied into the `segment_name` variable. If there is no currently open retained segment, `segment_name` is set to zero.

4.3. Temporary or Non-Retained Segments

Temporary segments are used for transient images. Temporary segments cannot be modified dynamically, and all portions of temporary segments which have already been drawn are deleted upon any new frame action. Primitives placed in temporary segments are not stored in the display list.

Create Temporary Segment

```
create_temporary_segment()
```

`create_temporary_segment()` creates a new, empty, nonretained or temporary, segment.

Close Temporary Segment

```
close_temporary_segment()
```

`close_temporary_segment()` closes the currently open temporary segment.

Get Temporary Segment Status

```
inquire_open_temporary_segment(open)
int *open;  /* Receives status of temporary segment */
```

`inquire_open_temporary_segment()` determines whether there is a currently open temporary segment. The *open* argument receives the status of whether there is a currently open temporary segment:

FALSE There is no currently open temporary segment.

TRUE There is a currently open temporary segment.

4.4. Saving and Restoring Segments on Disk

The two functions described in this section provide for saving segments on disk files and restoring segments from disk files. Only one segment is saved in a given file.

Save Segment on Disk File (SunCore Extension)

```
save_segment(segment_name, filename)
int segment_name;    /* Name of segment to save */
char *filename;      /* Pointer to a filename */
```

`save_segment()` saves the named retained segment on a specified disk file. Saved primitives are in NDC space. Dynamic segment attributes are also saved.

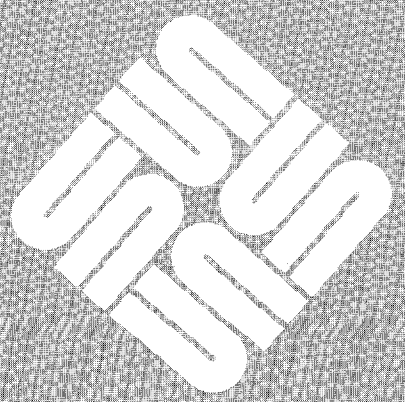
Restore Segment from Disk File (SunCore Extension)

```
restore_segment(segment_name, filename)
int segment_name;    /* Name of segment to create */
char *filename;      /* Pointer to a filename */
```

`restore_segment()` restores the named retained segment from a specified disk file. A new segment is created and the segment from the disk file is copied into it. The segment is then closed.

5

Output Primitives



5

Output Primitives

Output Primitives serve to describe objects in the world coordinate system. When the output primitive functions are called, *primitives* are placed in the currently open segment via drawing commands which eventually produce line and character output.

SunCore supports six kinds of output primitives, namely *moves*, *lines* and *polylines*, *polygons*, *text*, *markers* and *polymarkers*, and *rasters*. The table below summarizes these types of functions:

Table 5-1 *Summary of Output Primitive Functions*

Primitive	***Description***
Move	primitives alter the value of the current position (described below).
Line	primitives describe lines in world coordinates.
Polyline	primitives describe sequences of connected lines in world coordinates.
Polygon	primitives describe a closed polygon which will be filled with a color. The polygon primitives are a SunCore extension to the ACM Core specification.
Text	primitives describe character strings on the display.
Marker	primitives describe markers which are written on the display in a constant orientation, independent of any transformations which may be in effect.
Polymarker	primitives describe a sequence of markers which are written on the display in a constant orientation, independent of any transformations which may be in effect.
Rasters	primitive describes an array of one-bit or eight-bit pixels.

All primitive operations use world coordinates. Some of these operations affect the value known as the *current position*. The current position defines the current drawing location in the world coordinate system. *SunCore* maintains the value of the current position at all times. At initialization time, the current position is initialized to the origin of the world coordinate system.

In both 2 and 3D, coordinate positions can be specified in terms of absolute world coordinates, or coordinates can be specified relative to the current position.

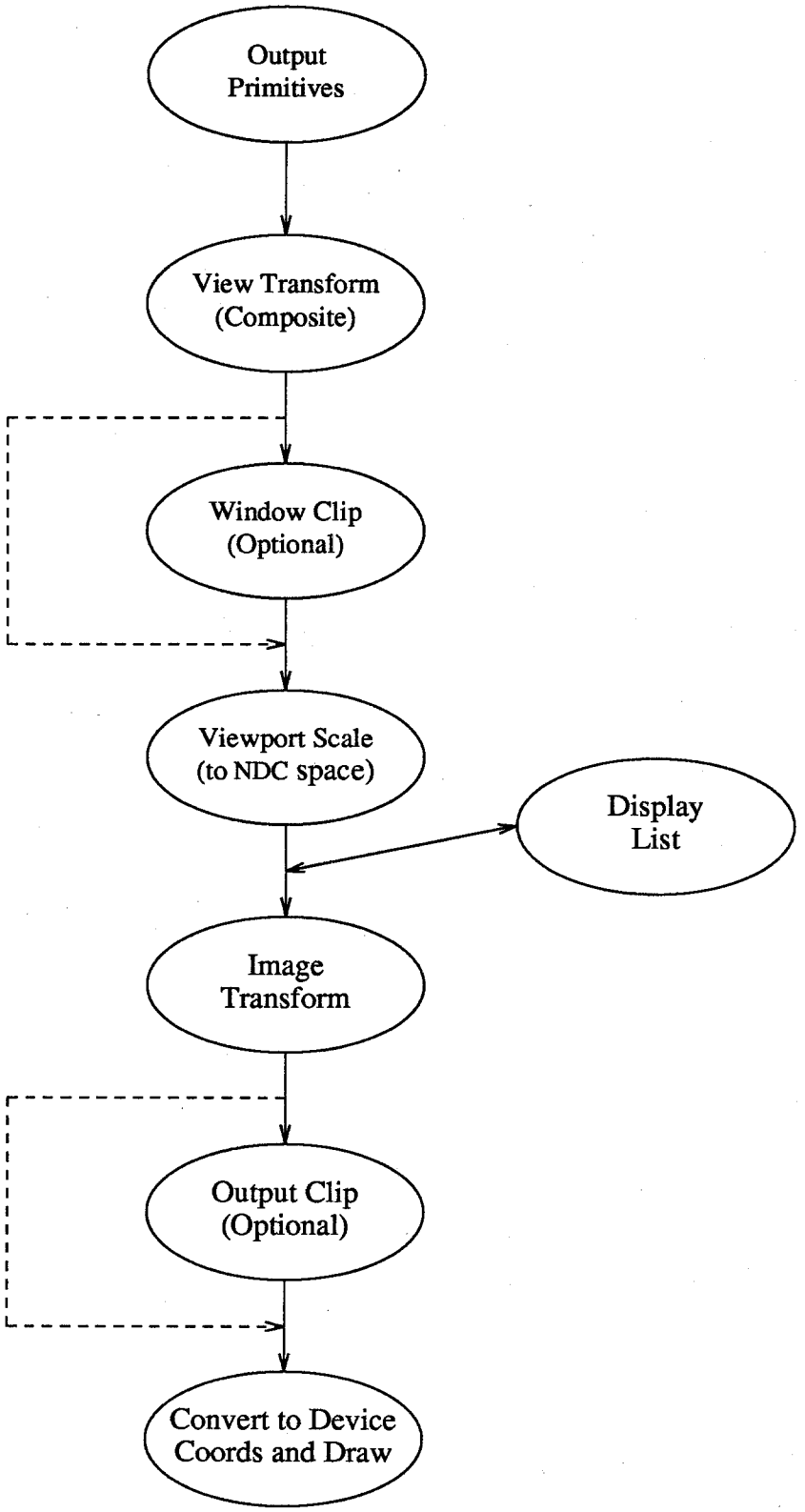
A segment must be open (see the `create_xxxxx_segment()` functions) before any output primitives may be used. A segment contains a set of output primitives which can subsequently be manipulated as a unit.

An output primitive is processed as follows:

1. The primitive is transformed to clipping coordinates using the *composite viewing transform.* This places the window boundaries at *umin*=–32767, *umax*=+32767, *vmin*=–32767, and *vmax*=+32767. The front clipping plane is at z=0 and the back clipping plane is at z=+32767.
2. The primitive is then clipped to the boundaries just mentioned if *window clipping* is enabled.
3. The output primitive is then *output scaled* to the viewport which is specified in NDC space.
4. The resulting primitive is then copied to the *display list* or *pseudo display file* (PDF) if the open segment is a retained segment.
5. Next, the primitive is transformed using the *image transform* which is an attribute of retained translatable or retained transformable segments.
6. The output primitive is then clipped again to the viewport boundaries if *output clipping* is enabled.
7. For each view surface which was selected when the segment was created, the primitive is then converted to physical device coordinates and drawn on the view surface.

If a change is made to certain dynamic segment attributes of a retained segment, the primitives in that segment are recovered from the PDF and used to erase the segment (if necessary) and redraw the segment following steps 5 through 7 above. The diagram below shows the above process in a graphical form.

Figure 5-1 *Flow Diagram of Output Primitive Processing*



Output primitives are drawn with the static primitive attributes set by the primitive attribute functions (see Chapter 6).

5.1. Moving the Current Position

There are four functions for moving the current position. `move_abs_2()` and `move_abs_3()` change the current position to an absolute position in world coordinates, whereas `move_rel_2()` and `move_rel_3()` change the current position by a delta relative to the current position.

Note that `move_abs_2()` and `move_rel_2()` are simply short forms of the corresponding 3D functions. The *z* coordinate of `move_abs_2()` is the *z* coordinate of the current position. The *z* delta of `move_rel_2()` is taken as zero.

Move to Absolute 2D Position

```
move_abs_2(x, y)
float x, y; /* x and y coordinates to move to */
```

`move_abs_2()` moves the current position to an absolute position. The current position is set to the values of *x* and *y* in 2D world coordinates. `move_abs_2()` only sets the current position; no drawing commands are output.

Move to Absolute 3D Position

```
move_abs_3(x, y, z)
float x, y, z;  /* x, y, and z coordinates to move to */
```

`move_abs_3()` moves the current position to an absolute position. The current position is set to the values of *x*, *y*, and *z* in 3D world coordinates. `move_abs_3()` only sets the current position; no drawing commands are output.

Move to Relative 2D Position

```
move_rel_2(dx, dy)
float dx, dy;   /* x and y coordinate deltas */
```

`move_rel_2()` increments the current position by the values given. The current position is set to the value of current position plus *dx* and *dy* in 2D world coordinates. `move_rel_2()` only sets the current position; no drawing commands are output.

Move to Relative 3D Position

```
move_rel_3(dx, dy, dz)
float dx, dy, dz;   /* x, y, and z coordinate deltas */
```

`move_rel_3()` increments the current position by the values given. The current position is set to the value of current position plus *dx*, *dy*, and *dz* in 3D world coordinates. `move_rel_3()` only sets the current position; no drawing commands are output.

5.2. Position Inquiry Functions

The position inquiry functions return the coordinates of the current position to the caller.

Inquire 2D Position

```
inquire_current_position_2(x, y)
float *x, *y;
```

`inquire_current_position_2()` returns the 2D world coordinates of the current position to the caller.

Inquire 3D Position

```
inquire_current_position_3(x, y, z)
float *x, *y, *z;
```

`inquire_current_position_3()` returns the 3D world coordinates of the current position to the caller.

5.3. Line Functions

The line functions draw lines on the currently selected *SunCore* view surfaces. Attributes of the line can be specified with additional calls to primitive attribute setting functions.

The primitive attributes of *line index*, *linestyle*, *linewidth*, and *pick_id* are applicable for lines.

- There is no open segment.

Describe Line in Absolute 2D Coordinates

```
line_abs_2(x, y)
float x, y;
```

`line_abs_2()` describes a line in 2D world coordinates. The line that `line_abs_2()` describes extends from the current position to the position specified by the *x* and *y* coordinates.

The current position is updated to the coordinates specified by *x* and *y*.

Describe Line in Absolute 3D Coordinates

```
line_abs_3(x, y, z)
float x, y, z;
```

`line_abs_3()` describes a line in 3D world coordinates. The line that `line_abs_3()` describes extends from the current position to the position specified by the *x*, *y*, and *z* coordinates.

The current position is updated to the coordinates specified by *x*, *y*, and *z*.

Describe Line in Relative 2D Coordinates

```
line_rel_2(dx, dy)
float dx, dy;
```

`line_rel_2()` describes a line in 2D world coordinates. The line that `line_rel_2()` describes extends from the current position to the position specified by the current position plus the *dx* and *dy* coordinates. The current position is updated by the deltas specified by *dx* and *dy*.

Describe Line in Relative 3D Coordinates

```
line_rel_3(dx, dy, dz)
float dx, dy, dz;
```

`line_rel_3()` describes a line in 3D world coordinates. The line that `line_rel_3()` describes extends from the current position to the position specified by the current position plus the *dx*, *dy*, and *dz* coordinates.

The current position is updated by the deltas specified by *dx*, *dy*, and *dz*.

5.4. Polyline Functions

The polyline functions describe connected sequences of lines. The first two or three arguments to a polyline function are arrays of the appropriate coordinates. Consider the polyline function:

```
polyline_abs_3(x_array, y_array, z_array, n)
float x_array[], y_array[], z_array[];
    /* x, y, and z arrays */
int n; /* Number of coordinates */
```

The sequence of lines that these arrays of coordinates describe starts at the current position, then draws to: (*x_array[0], y_array[0], z_array[0]*), then runs through the intermediate array values and ends at (*x_array[n-1], y_array[n-1], z_array[n-1]*) where *n* is the number of elements in each of the coordinate arrays. There are thus *n* lines in the figure described.

- The number of coordinates, *n*, is less than or equal to zero.
- There is no open segment.

Describe Line Sequence in Absolute 2D Coordinates

```
polyline_abs_2(x_array, y_array, n)
float x_array[], y_array[]; /* x and y coordinates */
int n;      /* number of array elements */
```

`polyline_abs_2()` describes a line sequence in absolute 2D world coordinates. The current position is updated to the end of the last line drawn.

Describe Line Sequence in Absolute 3D Coordinates

```
polyline_abs_3(x_array, y_array, z_array, n)
float x_array[], y_array[], z_array[];
    /* x, y, and z arrays */
int n; /* number of elements */
```

`polyline_abs_3()` describes a line sequence in absolute 3D world coordinates. The current position is updated to the end of the last line drawn.

Describe Line Sequence in Relative 2D Coordinates

```
polyline_rel_2(dx_array, dy_array, n)
float dx_array[], dy_array[]; /* x and y delta arrays */
int n; /* number of array elements */
```

`polyline_rel_2()` describes a line sequence in relative 2D world coordinates. The sequence of lines that this function describe starts at the current position, moves to: current position + *dx_array* [0], (dy_array [0]) then draws to: current position + *dx_array* [0], (dy_array [0]) + *dx_array* [1], (dy_array [1]) and so on. The current position is updated to the end of the last line drawn.

Describe Line Sequence in Relative 3D Coordinates

```
polyline_rel_3(dx_array, dy_array, dz_array, n)
float dx_array[], dy_array[], dz_array[];
    /* x, y, and z delta arrays */
int n;  /* number of elements */
```

`polyline_rel_3()` describes a line sequence in relative 3D world coordinates. The sequence of lines that this function describe starts at the current position, moves to: current position + *dx_array* [0], (dy_array [0], *dz_array [0]) then draws to: current position + (dx_array[0], dy_array[0], dz_array[0]) + (dx_array[1], dy_array[1], dz_array[1]) and so on. The current position is*

updated to the end of the last line drawn.

5.5. Text Functions

The functions described in the next section describe the text facilities available in *SunCore*. The inquiry functions that follow can be used to determine characteristics of text.

Draw Character String In World Coordinates

```
text(string);
char *string;
```

`text()` draws a character string in world coordinates. The character string specified by *string* is drawn from the current position. The current position is unchanged. The font, size, orientation, and so on, are set by calls to the set primitive attribute functions.

- There is no open segment.
- The character string contains one or more characters which cannot be drawn.
- The vectors that the current *charpath* and *charup* attributes describe are parallel.

5.6. Text Inquiry Functions

Text inquiry functions obtain the length that a character string would extend, in world coordinates, if the character string were actually drawn according to the current text primitive attributes.

- `inquire_text_extent_2()` was used to obtain the current position when `inquire_text_extent_3()` should have been used in order to avoid loss of information.
- The character string contains one or more characters which cannot be drawn.
- The vectors that the current *charpath* and *charup* attributes describe are parallel.

Inquire Text Extent 2

```
inquire_text_extent_2(string, dx, dy)
char *string;
float *dx, *dy;
```

`inquire_text_extent_2()` returns the extent of the character string specified by *string*, if the character string were drawn, unjustified, from the current position. The extent is returned in *dx* and *dy* in world coordinates relative to the current position.

The specified character string, and the values of the primitive attributes *font*, *charup*, *charsize*, *charpath*, *charspace*, and *charprecision* are used to calculate the vector which represents the extent of the character string.

In the current implementation of *SunCore*, this function only returns meaningful values if *charprecision* is CHARACTER.

Inquire Text Extent 3

```
inquire_text_extent_3(string, dx, dy, dz)
char *string;
float *dx, *dy, *dz;
```

`inquire_text_extent_3()` obtains the 3D extent, in world coordinates, of

the specified character string. `inquire_text_extent_3()` returns the extent of the character string specified by *string*, if the character string were drawn, unjustified, from the current position. The extent is returned in *dx*, *dy*, and *dz* in world coordinates relative to the current position.

The specified character string, and the values of the primitive attributes *font*, *charup*, *charsize*, *charpath*, *charspace*, and *charprecision* are used to calculate the vector which represents the extent of the character string.

In the current implementation of *SunCore*, this function only returns meaningful values if *charprecision* is CHARACTER.

5.7. Marker Functions

The *marker* functions place a character at a specific location on the display. The *polymarker* functions place a character at a sequence of locations on the display.

The marker character is any printable ASCII character, and is the value of the `marker_symbol` primitive attribute. The `marker_symbol` primitive attribute is set by the `set_marker_symbol()` function described in Chapter 6.

The markers are placed on the display without any of the rotations, translations, or scaling which is applied to text strings. Markers use the default orientation attributes.

- There is no open segment.

Plot Marker at Absolute 2D Coordinates

```
marker_abs_2(x, y)
float x, y; /* Absolute x and y Coordinates */
```

`marker_abs_2()` plots a marker at specified absolute 2D world coordinates. `marker_abs_2()` plots the marker at the absolute 2D coordinates specified by the *x* and *y* arguments. The current position is updated to be this point.

Plot Marker at Absolute 3D Coordinates

```
marker_abs_3(x, y, z)
float x, y, z;  /* Absolute x, y, and z Coordinates */
```

`marker_abs_3()` plots a marker at specified absolute 3D world coordinates. `marker_abs_3()` plots the marker at the absolute 3D coordinates specified by the *x*, *y*, and *z* arguments. The current position is updated to be this point.

Plot Marker at Relative 2D Coordinates

```
marker_rel_2(dx, dy)
float dx, dy;   /* x and y Coordinate Deltas */
```

`marker_rel_2()` plots the marker at the position relative to the current position, specified by the deltas *dx* and *dy*. The current position is updated to be this point.

Plot Marker at Relative 3D Coordinates

```
marker_rel_3(dx, dy, dz)
float dx, dy, dz;   /* x, y, and z Coordinate Deltas */
```

`marker_rel_3()` plots a marker at a specified relative 3D position. `marker_rel_3()` plots the marker at the position relative to the current position, specified by the deltas *dx*, *dy*, and *dz*. The current position is updated to be this point.

Plot Marker Sequence at Absolute 2D Coordinates

```
polymarker_abs_2(x_array, y_array, n)
float x_array[], y_array[]; /* Absolute x and y */
int n;  /* Number of Coordinates */
```

`polymarker_abs_2()` plots a sequence of markers at specified absolute 2D positions. `polymarker_abs_2()` plots a sequence of markers at the absolute positions specified by the *x_array* and *y_array* arguments. *n* specifies the number of coordinates in the arrays. The current position is updated to be the last point.

Plot Marker Sequence at Absolute 3D Coordinates

```
polymarker_abs_3(x_array, y_array, z_array, n)
float x_array[], y_array[], z_array[];
            /* Absolute x, y, and z */
int n;      /* Number of Coordinates */
```

`polymarker_abs_3()` plots a sequence of markers at specified absolute 3D positions. `polymarker_abs_3()` plots a sequence of markers at the absolute positions specified by the *x_array*, *y_array*, and *z_array* arguments. The number of coordinates in the array is given by the *n* argument. The current position is updated to be the last point.

Plot Marker Sequence at Relative 2D Coordinates

```
polymarker_rel_2(dx_array, dy_array, n)
float dx_array[], dy_array[];   /* x and y Deltas */
int n;  /* Number of Coordinates */
```

`polymarker_rel_2()` plots a sequence of markers at specified relative 2D positions. `polymarker_rel_2()` plots a sequence of markers at the positions relative to the current position, specified by the deltas *dx_array* and *dy_array*. The number of deltas in the arrays is specified by *n*. The current position is updated to be the last point.

Plot Marker Sequence at Relative 3D Coordinates

```
polymarker_rel_3(dx_array, dy_array, dz_array, n)
float dx_array[], dy_array[], dz_array[];
        /* x, y, and z Deltas */
int n;  /* Number of Coordinates */
```

`polymarker_rel_3()` plots a sequence of markers at specified relative 3D positions. `polymarker_rel_3()` plots a sequence of markers at the positions relative to the current position, specified by the deltas *dx_array*, *dy_array*, and *dz_array*. The number of deltas in the arrays is specified by *n*. The current position is updated to be the last point.

5.8. 3D Polygon Shading Parameters (SunCore Extension)

When drawing 3D polygons on the Sun color displays, several shading options are available. The functions described in this section provide shading control. These shading parameters may be changed at any time and are not stored in the display list. Therefore a segment may be drawn with fast shading at one time, and then drawn again later with smooth shading.

Set Shading Parameters

```
set_shading_parameters(ambient, diffuse,
    specular, flood, bump, hue, style)
float ambient;  /* percent background light */
float diffuse;  /* percent diffuse reflection */
float specular; /* percent specular reflection */
float flood;    /* percent flood lighting */
float bump; /* specular power 2 .. 9 */
int hue;    /* color index range to generate */
        /* 0 = 0 .. 255, 1 = 0 .. 63 */
        /* 2 = 64 .. 127, 3 = 128 .. 191 */
        /* 4 = 192 .. 255 */
int style;  /* Type of surface shading to do: */
        /* CONSTANT, GOURAUD, PHONG */
```

`set_shading_parameters()` specifies the parameters for rendering 3D polygons on the color display. See `set_polygon_interior_style()` for the ways in which these shading parameters are used. CONSTANT style shading gives constant intensity over the polygon using the color set by `set_fill_index()`. GOURAUD style shading linearly interpolates between vertices where the intensity at each vertex is set by the `set_vertex_indices()` function. PHONG style shading produces smooth shading using the other parameters (only with convex polygons).

The equation used for PHONG style shading is:

$$pixelshade = ambient + diffuse\,(L \bullet N) + specular\,(H \bullet N)^{bump} - (flood * z)$$

where L is the direction vector of the light source, N is the surface normal vector, H is a vector which is the average of L and E (the eye direction vector), and z is depth in NDC.

Here are some useful sets of PHONG parameters:

Table 5-2 *2 Useful PHONG Parameters*

Parameter	*Value*	*Value*
ambient	0.05	0.05
diffuse	0.94	0.74
specular	0.0	0.20
flood	0.0	0.0
bump	0.0	7.0
hue	0	0

Specify Direction of Light Source

```
set_light_direction(dx, dy, dz)
float dx, dy, dz;
```

`set_light_direction()` specifies the direction of the light source from the object. This assumes NDC space where the direction from object to viewer is always {0.0, 0.0, –1.0}. Hence, to place the light source at the viewer, the light direction is (0.0, 0.0, –1.0). The light direction vector is only used if the shading style is GOURAUD or PHONG. A useful light direction is (–0.2, 0.2, –1.0).

Set Vertex Normals

```
set_vertex_normals(xlist, ylist, zlist, n)
float xlist[], ylist[], zlist[];
int n;
```

`set_vertex_normals()` sets the surface normal vectors for each vertex of the subsequent 3D polygon primitives (`polygonabs_3()` or `polygonrel_3()`). These normals are used for PHONG style shading. For GOURAUD style shading, use `set_vertex_indices()`. The number of elements in the list, *n*, must be equal to the number of vertices in the subsequent call to `polygon`*xxx*`_3()`.

Set Vertex Indices

```
set_vertex_indices(color_index_list, n)
int color_index_list[];
int n;
```

`set_vertex_indices()` specifies a color index for each vertex of the next `polygon`*xxx*`_3()` primitive. GOURAUD shading linearly interpolates these color indices for smooth shading in the interior of the polygon. The number of elements in the list, *n*, must be equal to the number of vertices in the subsequent call to `polygon`*xxx*`_3()`.

Note: If the *hue* argument to `set_shading_parameters()` is 0, then *color_index_list* is an index into the predefined colormap. If *hue* is 1, then the first 64 values in the predefined colormap are interpolated into *color_index_list*. If *hue* is 2, then the second 64 values are used, and so on.

Set Z Buffer Cut

```
set_zbuffer_cut(surface_name, xlist, zlist, n)
struct vwsurf *surface_name;    /* See Appendix B */
float xlist[], zlist[];
int n;
```

`set_zbuffer_cut()` specifies a cutaway view of 3D polygon objects when hidden surfaces are being removed. `set_zbuffer_cut()` specifies an array of depths in NDC space. Any parts of objects which are closer to the viewer than this piecewise-linear function are clipped away.

Note: this function has no effect on Graphics Processor view surfaces, i.e. `gp1dd` or `gp1pixwindd`. *xlist* is assumed to be monotonically increasing. This function specifies a piecewise-linear cutaway threshold in the *z* coordinate, which, given any *x* coordinate, is constant in *y*. The default cutaway depth is 0 for all values of *x*. Values of *x* less than *xlist* [0] or greater than *xlist* [*n* - 1] will have the default depth. The view surface must have been initialized with the *hidden* flag on.

5.9. Polygon Functions (SunCore Extension)

The polygon functions are a *SunCore* extension to the ACM Core specification. The polygon functions describe connected sequences of lines which form closed figures. The polygons are filled in with color as specified by the `set_fill_index()` primitive attribute, or are shaded according to the current shading parameters, depending on the `polygoninterior_style` primitive attribute. Only polygons created by the 3D polygon functions may be shaded.

The first two or three arguments to a polygon function are arrays of the appropriate coordinates. Consider the polygon function:

```
polygon_abs_3(x_array, y_array, z_array, n)
float x_array[], y_array[], z_array[];
    /* x, y, and z coordinates */
int n;  /* Number of coordinates */
```

The bounding sequence of edges that these arrays of coordinates describe pass from the first point *x_array[0],(y_array[0], z_array[0])*, then runs through the intermediate array values to (*x_array[n-1], y_array[n-1], z_array[n-1]*) and then back to the first point. *n* is the number of elements in each of the coordinate arrays. There are thus *n* sides in the figure described.

Note that the polygon functions describe a closed figure. The last coordinate in the array of points is connected to the first point.

- The number of coordinates, *n*, is less than or equal to two.
- There is no open segment.

Describe Polygon in Absolute 2D Coordinates

```
polygon_abs_2(x_array, y_array, n)
float x_array[], y_array[]; /* x and y coordinates */
int n;  /* number of array elements */
```

`polygon_abs_2()` describes a polygon in absolute 2D world coordinates. The current position is set to the first point.

Describe Polygon in Absolute 3D Coordinates

```
polygon_abs_3(x_array, y_array, z_array, n)
float x_array[], y_array[], z_array[];
    /* x, y, and z coordinates */
int n;  /* number of array elements */
```

`polygon_abs_3()` describes a polygon in absolute 3D world coordinates. The current position is set to the first point.

Describe Polygon in Relative 2D Coordinates

```
polygon_rel_2(dx_array, dy_array, n)
float dx_array[], dy_array[];   /* x and y deltas */
int n;  /* number of array elements */
```

`polygon_rel_2()` describes a polygon in relative 2D world coordinates. The first array value specifies a displacement from the current position; remaining array values specify displacements from the preceding point. The current position is set to the first point.

Describe Polygon in Relative 3D Coordinates

```
polygon_rel_3(dx_array, dy_array, dz_array, n)
float dx_array[], dy_array[], dz_array[];
        /* x, y, and z deltas */
int n;  /* number of array elements */
```

`polygon_rel_3()` describes a polygon in relative 3D world coordinates. The first array value specifies a displacement from the current position; remaining array values specify displacements from the preceding point. The current position is set to the first point.

5.10. Raster Primitive Functions (SunCore Extension)

The raster primitive functions described in the following sections allow the *SunCore* application program to access and manipulate rectangular arrays of pixels. Both monochrome and color frame raster primitives are supported. These functions are not a part of the standard Core system.

Raster Output Primitive

```
put_raster(raster)
struct suncore_raster *raster;
```

`put_raster()` draws a rectangular 1-bit or 8-bit deep raster and enters it into the current segment. The raster may not be used in transformable segments, because rasters cannot be scaled or rotated in the current release of *SunCore*. A raster primitive may, however, be picked or dragged if it is entered in a translatable segment. The current position is at the lower left-hand corner of the raster.

Note that `put_raster()` is *device dependent* in that it is written to the right and upward from the current position a specified number of PIXELS in height and width. The current position is unchanged.

Here is the definition of the `suncore_raster` structure.

```
struct suncore_raster {
    int width;
    int height;
    int depth;
    short *bits;
};
```

The *depth* parameter can be 1 or 8 bits per pixel.

The bits of the raster are stored in the following order for*depth* = 1: The first word is the upper left 16 horizontal bits, with the high order bit being the leftmost bit. The first (*width* + 15)/16 words comprise the top row of the rectangle. The number of words of storage that *bits* points to is:

```
((width+15) / 16) * height
```

for *depth* = 1.

Rasters of *depth* = 8 are stored as successive bytes in row order. The number of bytes that *bits* points to is:

```
width * height
```

for *depth* = 8.

If a 1-bit deep raster is written to a color view surface, '0' bits select the background color and '1' bits select the color specified by the *fill index* primitive attribute.

Note that output clipping is always done on raster primitives.

Read Raster from Monochrome or Color Frame Buffer

```
get_raster(surface_name, xmin, xmax,
    ymin, ymax, x, y, raster)
struct vwsurf *surface_name; /* See Appendix B */
float xmin, ymin, xmax, ymax;
    /* Region of NDC space to read */
int x, y;   /* starting point pixel offsets
    in raster relative top left */
struct suncore_raster *raster;  /* Returned Raster */
```

`get_raster()` reads a specified region of the monochrome or color frame buffer into a storage area. `get_raster()` requires an area of memory large enough to hold the raster region that it returns. It is the user's responsibility to allocate this storage area before calling `get_raster()`. The `size_raster()` and `allocate_raster()` functions may be used to do this:

```
size_raster(surface_name, xmin, xmax, ymin, ymax, &raster);
allocate_raster(&raster);
if (raster.bits == NULL)
    error case - the raster could not be allocated
else
    continue with the processing
```

To free the area when finished with the raster, call the `free_raster()` function:

```
free_raster(&raster);
```

Hence, a large raster may be allocated and then portions of it filled with data using `get_raster()` with various *x*,.I y offsets, in pixel coordinates from the top left hand corner of the raster.

Set Size of Raster in NDC

```
size_raster(surface_name, xmin, xmax, ymin, ymax, raster)
struct vwsurf *surface_name;
float xmin, xmax, ymin, ymax;
struct suncore_raster *raster;
```

`size_raster()` returns the raster with the pixel coordinates *width*, *height*, and *depth*, for a specified region of NDC space and a specified view surface. On return, `raster.bits` is set to NULL.

Allocate Space for a Raster

```
allocate_raster(raster)
struct suncore_raster *raster;
```

Given a raster whose *width*, *height*, and *depth* fields were filled by the `size_raster()` function (described above), `allocate_raster()` allocates the memory required for that raster and sets the `raster.bits` pointer. `allocate_raster()` returns a NULL pointer value in `raster.bits` if it is unable to obtain enough memory for the raster structure.

Free Space of a Raster

```
free_raster(raster)
struct suncore_raster *raster;
```

`free_raster()` frees the memory used by a specified raster, if `raster.bits` is not NULL.

Copy a Raster to a Disk Raster File

```
raster_to_file(raster, map, fd, replicate)
struct suncore_raster *raster;
struct {
    int type; /* 1 for RGB color table */
    int nbytes; /* 3 times number
        of color table elements */
    char *data; /* ptr to nbytes/3 red,
        blue, and green bytes */
} *map;
int fd; /*standard file descriptor for C programs */
    /* FORTRAN logical unit number
        for FORTRAN programs */
    /* Pascal file variable for Pascal programs */
int replicate;  /* magnification factor */
```

`raster_to_file()` copies a raster to a disk file in Sun's standard raster file format. If `map.nbytes` = 0, no color map data will be written. This would normally be the case for rasters copied from the bitmap display.

The *replicate* parameter specifies whether the raster should be magnified on transmission to the file. The raster is transmitted without magnification if and*replicate*= 1, pixel-replication zoom for a factor of 2 magnification if *replicate*= 2.

Note: The colormap information provided to `raster_to_file()` includes integer color values in the range 0 – 255. *SunCore* normally takes floating point color values in the range 0 – 1.0.

The format of the generated disk file can be found in the include file in `<rasterfile.h>`. Disk raster files can be printed on a raster addressable hard copy device by using the `lpr`(1) command with the –v option.

Get a Raster from a Disk File

```
file_to_raster(fd, raster, map)
int fd;
    /* standard file descriptor for C programs */
    /* Fortran logical unit number for Fortran programs */
    /* Pascal file variable for Pascal programs */
struct suncore_raster *raster;
struct {
    int type; /* 1 for RGB color table */
    int nbytes; /* 3 times number
        of color table elements */
    char *data; /* ptr to nbytes/3 red,
        blue, and green bytes */
} *map;
```

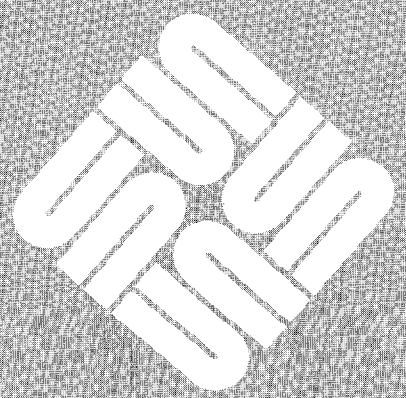
`file_to_raster()` allocates enough memory for a raster stored on a disk file, then fills in all fields of the raster and map structures. Note that this function frees `map.data`, unless *data* is NULL, and allocates `map.data` each time it is called — therefore `map.data` is only valid in the last call to this function. The `raster.bits` field is set to NULL if there is not enough room to allocate the

raster.

The format of the disk file can be found in the include file in `<rasterfile.h>`.

6

Attributes



6

Attributes

Attributes in *SunCore* specify general characteristics for segments and for output primitives.

There are two major divisions of attributes. One set of attributes is called *segment attributes* and applies only to retained segments. The other set is called *primitive attributes* and applies only to output primitives. There are no attributes which apply to both retained segments and to output primitives.

Attributes are further subdivided into *static* and *dynamic*. Static attributes specify characteristics of retained segments or output primitives which apply for the entire lifetime of those objects. Dynamic attributes specify characteristics of segments which can change during the lifetime of those segments. Static primitive attributes are stored in the display list so that subsequent manipulation of a segment is performed with the appropriate attributes.

6.1. Primitive Static Attributes

The list below defines the primitive static attribute values.

line index
: is an index into three `float` arrays which determine the red, green, and blue components of the color displayed for line and polyline output primitives. Index value 0 corresponds to the background color. For lines and polylines on monochrome displays, a non-zero *line index* gives black lines on a white background. *SunCore* initializes *line index* to 1. The range of possible values is 0 to 255.

fill index
: is an index into three `float` arrays which determine the red, green, and blue components of the color displayed for polygon and raster output primitives. Index value 0 corresponds to the background color. For monochrome displays, the values form a set of definitions for texture, described later. *SunCore* initializes *fill index* to 1. The range of possible values is 0 to 255.

text index
: is an index into three `float` arrays which determine the red, green, and blue components of the color displayed for markers and text. Index value 0 corresponds to the background color. For text and markers on monochrome displays, a non-zero *text*index" gives black on a white background. *SunCore* initializes *text index* to 1. The range of possible values is 0 to 255.

linestyle
is an `int` value which controls the appearance of lines drawn. *Linestyle* can assume the values:

SOLID	Solid lines,
DOTTED	Dotted lines,
DASHED	Dashed lines,
DOTDASHED	Dotdashed lines.

The definitions of these constants can be found in `<usercore.h>`. *SunCore* sets *linestyle* to SOLID at initialization time.

polygon interior style
is an `int` value which controls the interior filling style for polygons. *polygon interior style* can have the values:

PLAIN	Solid color polygon
SHADED	Shading style is set dynamically by `set_shading_parameters()`. Only 3D polygons may be shaded.

SunCore sets *polygon interior style* to PLAIN at initialization time.

polygon edge style
is not implemented in the current release of *SunCore*.

linewidth
is a `float` value which describes, in world coordinates, the width of drawn lines. *SunCore* sets *linewidth* to 0.0 (the minimum) at initialization time.

pen
is an `int` value which is passed to the device driver to select a particular device dependent pen. *SunCore* initializes *pen* to 0.

font
is an `int` value which determines the character font in which text will be written. *Font* can assume the following values (for *charprecision*=CHARACTER):

ROMAN	If *charprecision*=STRING, this gives a large raster font.
GREEK	If *charprecision*=STRING, this gives the default raster font.
SCRIPT	If *charprecision*=STRING, this gives a small raster font.
OLDENGLISH	If *charprecision*=STRING, this is equivalent to a bold version of GREEK.
STICK	If *charprecision*=STRING, this is equivalent to a medium sized ROMAN raster font.
SYMBOLS	Currently holds some electronics symbols (character values 32 through 47). If *charprecision*=STRING, this is

equivalent to a bold version of STICK.

SunCore sets *font* to STICK at initialization time.

charsize
is a pair of `float` values which determine the size of characters, in world coordinates. *SunCore* sets the default character width to 11.0 and the default character height to 11.0 at initialization time.

charup
attribute consists of three `float` values which represent a vector giving the direction of 'up' for characters:

```
(dx_charup, dy_charup, dz_charup)
```

in world coordinates. Usually, *charup* is normal to *charpath*. *SunCore* establishes the default as a vector in the positive *y* direction (0.0, 1.0, 0.0) at initialization time.

charpath
consists of three `float` values which represent a vector:

```
(dx_charpath, dy_charpath, dz_charpath)
```

that determines the direction, in world coordinates, in which character strings will extend. *SunCore* sets the *charpath* attribute to (1.0, 0.0, 0.0) at initialization time.

charspace
is a single `float` value specifying the space, in world coordinates, which should be inserted between characters in a text string. *SunCore* establishes *charspace* with the value 0.0 at initialization time.

charjust
is not implemented in the current release of *SunCore*.

charprecision
is an `int` value which controls the quality of the text drawing operation. *Charprecision* can have the values:

STRING	Fast raster fonts, fixed size, and fixed orientation.
CHARACTER	Hershey vector fonts.

marker symbol
determines the character which is plotted on the displays by the marker and polymarker functions described in Chapter 5. Any printable ASCII character can be used as the marker character.

Note: The ACM Core specifies that the integer values 1 through 5 represent specific characters. *SunCore* does *not* implement this feature.

pick id
is an `int` value identifying the next output primitive. The input primitives use this number for user interaction with segments and primitives within segments.

rasterop
specifies the rasterop used when writing to the display. It can be one of:

NORMAL Source value is written to the display.

XORROP Source value is exclusive or'ed with the value already in the display before being written to the display.

ORROP Source value is or'ed with the value already in the display before being written to the display.

This attribute is ignored if `set_drag()` was specified as TRUE.

The functions listed in the subsections below each set the specified attribute value for the indicated primitive attribute.

- One or more of the attribute values is incorrect.
- No character orientation can be established because *dx_charpath*, *dy_charpath*, and *dz_charpath* are all zero.
- No character up direction can be established because *dx_charup*, *dy_charup*, and *dz_charup* are all zero.

6.2. Using Texture for Color Attributes on the Monochrome Display

When a monochrome display is used, the *fill index* attribute is used to determine how a region of the screen is textured when using the polygon output primitives. Texturing is done in terms of 16 × 16 pixel regions of the screen. There are 16 rows of 16 pixels each. The *fill index* attribute selects an entry from each of three arrays of `float` values in the range 0.0 through 1.0, representing red, green, and blue. In the case of the monochrome display, each of these three `float` numbers is converted to an integer between 0 and 255. Each of the 8-bit numbers is divided into two four-bit quantities, which we can call A and B.

Table 6-1 *Structure of a Fill-Index Value*

Red		***Green***		***Blue***	
Select B	*Select* A	*Length* B	*Length* A	*Rotate* B	*Rotate* A

Select A and *Select B* are four-bit values which are used to select an *A pattern* and a *B pattern* out of the table of numbers shown below.

Table 6-2 *Texture Selection Values*

Four-Bit Value	*Hexadecimal Pattern*	*Binary Pattern*
0	0000	0000000000000000
1	8000	1000000000000000
2	8080	1000000010000000
3	8410	1000010000010000
4	8888	1000100010001000
5	9124	1001000100100100
6	9494	1001010010010100
7	A552	1010010101010010
8	AAAA	1010101010101010
9	EB6E	1110101101101110
10	DDDD	1101110111011101
11	F7F7	1111011111110111
12	FFFF	1111111111111111
13	E3E3	1110001111100011
14	FF00	1111111100000000
15	00FF	0000000011111111

The patterns are then laid down in the texture field, pixels, as described in the pseudo code below.

```
let x = y = Pattern A
for index = 0 to Length A - 1
    pixels[index] = x | y
    if Rotate A & 1 then rotate x one bit right
    if Rotate A & 2 then rotate x one bit left
    if Rotate A & 4 then rotate y one bit right
    if Rotate A & 8 then rotate y one bit left

let x = y = Pattern B
for index = Length A to Length A + Length B - 1
    pixels[index] = x | y
    if Rotate B & 1 then rotate x one bit right
    if Rotate B & 2 then rotate x one bit left
    if Rotate B & 4 then rotate y one bit right
    if Rotate B & 8 then rotate y one bit left
```

If the value of

```
length A + length B
```

is less than 16, the processes described above are repeated as many times as required to fill the 16 line region.

The above encoding provides for an enormous number of textures. Here are a few of the useful ones.

Table 6-3 *Useful Texture Selection Values*

Texture	*Red*	*Green*	*Blue*
Hatched Left	0.1334	0.5020	0.3529
Hatched Right	0.1334	0.5020	0.6471
Wallpaper	0.4667	0.5334	0.2118
Black	0.0000	0.2667	0.3882
White	0.2667	0.4001	0.8001
Wavy Lines	0.3334	0.4001	0.1334
Grey Tone	0.5334	0.4001	0.5334
Cross Hatched	0.5334	0.4001	0.1334

Assign Colors to Indices

```
define_color_indices(surface_name, i1, i2,
    red_array, green_array, blue_array)
struct vwsurf *surface_name;    /* See appendix B */
int i1, i2; /* indices range from 0 through 255 */
float red_array[], green_array[], blue_array[];
```

`define_color_indices()` defines entries in the color lookup table of a view surface. The three arrays provide the values for red, green, and blue respectively. The value of each element in the color arrays is in the range 0.0 through 1.0. The function defines all the indices in the color index table between *i1* and *i2* inclusive, using the first *i2-i1+1* elements from each of the three arrays.

Subsequent calls to the `set_xxx_index()` function selects a color from the lookup table to use as a color attribute.

Location 0 in the color tables is the background color for the view surface. For the monochrome displays, lines, text, and markers are drawn black for any color index other than 0.

SunCore initializes the lookup table for monochrome view surfaces such that for the ith entry, red[i] = i, green[i] = 255−i, and blue[i] = i. *SunCore* initializes color view surfaces which have a full 256-element lookup table such that entry 0 is gray, entry 1 is black, entries 2 through 63 contain an intensity ramp in red, entries 64 through 127 contain an intensity ramp in green, entries 128 through 191 contain an intensity ramp in blue, and entries 192 through 255 contain an intensity ramp in yellow (red+green). See appendix B for details of color view surfaces with fewer than 256 entries in the lookup table.

Note: If the *SunCore* application is run in the *SunView* environment, `vwsurf.cmapname` and `vwsurf.cmapsize` *must* be defined in order to cooperate with colormap sharing provided by *SunView*.

Select a Line Color Attribute

```
set_line_index(index)
int index;  /* range 0 through 255 */
```

`set_line_index()` selects a color by providing an index into the tables defined by the `define_color_indices()` function. This color attribute is applied to subsequent line and polyline output primitives.

Select a Polygon and Raster Color

```
set_fill_index(index)
int index;  /* range 0 through 255 */
```

`set_fill_index()` selects a color by providing an index into the tables defined by the `define_color_indices()` function. This color attribute is applied to subsequent polygon and raster output primitives.

Select a Text and Marker Color

```
set_text_index(index)
int index;  /* range 0 through 255 */
```

`set_text_index()` selects a color by providing an index into the tables defined by the `define_color_indices()` function. This color attribute is applied to subsequent text and marker output primitives.

Set Linewidth

```
set_linewidth(linewidth)
float linewidth; /* unit of width
    is 1 percent of NDC space */
```

`set_linewidth()` specifies the *linewidth* attribute for the output primitives. *SunCore* initializes *linewidth* to 0.0, which results in a one pixel wide line.

If XOR'ing is enabled (via the `set_rasterop()` or `set_drag()` functions), lines whose pixel width is greater than one may partially overwrite themselves, resulting in poorly drawn wide lines. Redrawing the lines with XOR'ing off will draw the lines correctly (until this problem is fixed).

Set Linestyle

```
set_linestyle(linestyle)
int linestyle;  /* SOLID, DOTTED, */
                /* DASHED, DOTDASHED */
```

`set_linestyle()` specifies the *linestyle* attribute for output primitives. *SunCore* initializes *linestyle* to SOLID.

Select Plain or Shaded Polygons

```
set_polygon_interior_style(style)
int style;  /* PLAIN, SHADED */
```

`set_polygon_interior_style()` specifies the method of filling for polygons. If the filling method is SHADED, polygons are shaded according to the parameters set by the `set_shading_parameters()` function. Only 3D polygons may be shaded.

Set Polygon Edge Style (No Effect)

```
set_polygon_edge_style(style)
int style;  /* SOLID, INTERIOR */
```

`set_polygon_edge_style()` specifies the method of drawing the edges of a polygon. This function has no effect in the current release of *SunCore*.

Set Font

```
set_font(font)
int font;   /* ROMAN, GREEK, SCRIPT */
            /* OLDENGLISH, STICK, SYMBOLS */
```

`set_font()` specifies the *font* attribute for the output primitives. *SunCore* initializes *font* to STICK. If the *charprecision* attribute is set to STRING, ROMAN

gives a small Roman font, GREEK gives a stick figure font, SCRIPT gives a tiny stick figure font, OLDENGLISH gives a bold version of GREEK, STICK gives a medium sized ROMAN raster font, and SYMBOLS gives a bold version of STICK. The STRING precision fonts are 'raster' fonts and are not scalable or rotatable, hence they are in pixel coordinates and are larger on the color surface than on the monochrome bitmap display.

Select a Device Dependent Pen (no effect)

```
set_pen(pen)
int pen;
```

This function has no effect on the standard *SunCore* view surfaces.

Set Character Size

```
set_charsize(charwidth, charheight)
float charwidth, charheight;
```

`set_charsize()` specifies the *charsize* attribute for the text output primitive, in world coordinates. If the *charprecision* attribute is set to STRING, `set_charsize()` has no effect, except to control the target extent of the text for the `await_pick()` function. If the *charprecision* attribute is set to CHARACTER, `set_charsize()` sets the average size of a character, given that each character has its own size.

Define Character Spacing for Output Primitives

```
set_charspace(charspace)
float charspace;
```

`set_charspace()` specifies the *space* attribute for the text output primitive, in world coordinates. It is used to insert additional space between characters in text strings. If the *charprecision* attribute is set to STRING, `set_charspace()` has no effect.

Set Character Up Vector 2

```
set_charup_2(dx, dy)
float dx, dy;
```

`set_charup_2()` specifies the *charup* attribute for the text output primitive, in world coordinates. Note that the *dz* offset is set to 0.0 for this function. If the *charprecision* attribute is set to STRING, `set_charup_2()` has no effect; otherwise it specifies the *upward* direction for the characters. This provides for slanting, mirror imaging, and so on, for characters.

Set Character Up Vector 3

```
set_charup_3(dx, dy, dz)
float dx, dy, dz;
```

`set_charup_3()` specifies the *charup* attribute for the text output primitive, in world coordinates. If the *charprecision* attribute is set to STRING, `set_charup_3()` has no effect; otherwise it specifies the direction of *upward* for the characters. This provides for slanting, mirror imaging and such, for characters.

Set Character Path 2

```
set_charpath_2(dx, dy)
float dx, dy;
```

`set_charpath_2()` specifies the *charpath* attribute for the text output

primitive, in world coordinates. Note that the *dz* offset is set to 0.0 for this function. If the *charprecision* attribute is set to STRING, `set_charpath_2()` has no effect; otherwise the character string is written in this direction.

Set Character Path 3

```
set_charpath_3(dx, dy, dz)
float dx, dy, dz;
```

`set_charpath_3()` specifies the *charpath* attribute for the text output primitive, in world coordinates. If the *charprecision* attribute is set to STRING, `set_charpath_3()` has no effect; otherwise the character string is written in this direction.

Specify Text Justification (No Effect)

```
set_charjust(just)
int just;
```

`set_charjust()` specifies how text strings should be justified. This function has no effect in the current release of *SunCore*.

Set Character Precision

```
set_charprecision(charprecision)
int charprecision;  /* STRING, CHARACTER */
```

`set_charprecision()` selects the method of drawing text.

STRING	Specifies characters of fixed size and orientation, which are drawn rapidly using raster operations. This is the default.
CHARACTER	Specifies Hershey vector fonts, which can be clipped and transformed.

Set Marker Symbol

```
set_marker_symbol(marker)
int marker; /* Character to use as Marker - 32 .. 127 */
```

`set_marker_symbol()` establishes the *marker symbol* primitive attribute. The character specified by the *marker* argument in the `set_marker_symbol()` function call is subsequently used as the marker character by the marker and polymarker functions.

Set Pick ID

```
set_pick_id(pick_id)
int pick_id;
```

`set_pick_id()` specifies the *pick id* attribute for output primitives. The *pick id* attribute is only used by the *await_pick* input function. Subsequent output primitives are identified by the specified *pick id* when they are detected by the mouse pointing device, via the `await_pick()` input function.

Select Rasterop to Display Memory (SunCore Extension)

```
set_rasterop(rop)
int rop;     /* XORROP, ORROP, NORMAL */
```

`set_rasterop()` selects Xor'ing or or'ing of primitives to display memory.

Specify All Primitive Attributes

```
set_primitive_attributes(attributes)
struct {
    int lineindx, fillindx, textindx;
    int linestyl, polylinestyl, polyedgestyl;
    float linwidth;
    int pen, font;
    float charwidth, charheight;
    float charupx, charupy, charupz, charupw;
    float charpathx, charpathy, charpathz, charpathw;
    float charspacex, charspacey, charspacez, charspacew;
    int chjust, chquality;
    int marker, pickid, rasterop;
} *attributes;
```

`set_primitive_attributes()` is a composite function which provides a means to set all the primitive attributes in a single function call. Note that the function call:

```
set_primitive_attributes(&PRIMATTS)
```

sets all the primitive attributes to their default values. PRIMATTS is defined in `<usercore.h>`.

6.3. Inquiring Primitive Static Attribute Values

The functions described in the sections that follow allow the user to inquire static attribute values of the *SunCore* primitives.

- A 2D inquiry function was used when a 3D inquiry function should have been used to avoid loss of information.

Inquire Color Indices

```
inquire_color_indices(surface_name, i1, i2,
    red_array, green_array, blue_array)
struct vwsurf *surface_name;    /* See appendix B */
int i1, i2;     /* Start and end table indices */
float red_array[];  /* Range is 0.0 thru 1.0 */
float green_array[];    /* Range is 0.0 thru 1.0 */
float blue_array[]; /* Range is 0.0 thru 1.0 */
```

`inquire_color_indices()` obtains the color lookup table for the specified view surface. *surface_name* is the name of the view surface for which the color lookup tables should be obtained.

`inquire_color_indices()` takes entries from the color lookup tables, starting at index *i1* (relative to zero) and ending at index *i2*. The color lookup tables for a given color are stored in

```
array[0] through array[i2-i1]
```

Inquire Line Index

```
inquire_line_index(index)
int *index;
```

`inquire_line_index()` obtains the current color index for coloring line and polyline output primitives.

Inquire Fill Index

```
inquire_fill_index(index)
int *index;
```

`inquire_fill_index()` obtains the current color index for coloring polygon and raster output primitives.

Inquire Text Index

```
inquire_text_index(index)
int *index;
```

`inquire_text_index()` obtains the current color index for coloring marker and text output primitives.

Inquire Linewidth

```
inquire_linewidth(linewidth)
float *linewidth;
```

`inquire_linewidth()` obtains the *linewidth* attribute, in percent of NDC space, for the output primitives.

Inquire Linestyle

```
inquire_linestyle(linestyle)
int *linestyle; /* SOLID, DOTTED, */
                /* DASHED, DOTDASHED */
```

`inquire_linestyle()` obtains the *linestyle* attribute for the output primitives.

Obtain Polygon Shading Method

```
inquire_polygon_interior_style(style)
int *style; /* PLAIN, SHADED */
```

`inquire_polygon_interior_style()` obtains the method of filling for polygons.

Inquire Polygon Edge Style

```
inquire_polygon_edge_style(style)
int *style; /* SOLID, INTERIOR */
```

`inquire_polygon_edge_style()` obtains the current method of drawing polygon edges.

Inquire Pen

```
inquire_pen(pen)
int *pen;   /* Device dependent pen selector */
```

`inquire_pen()` obtains the *pen* attribute for the text output primitive.

Inquire Font

```
inquire_font(font)
int *font;  /* ROMAN, GREEK, SCRIPT, OLDENGLISH, */
            /* STICK, SYMBOLS */
```

`inquire_font()` obtains the *font* attribute for the text output primitive.

Inquire Character Size

```
inquire_charsize(charwidth, charheight)
float *charwidth, *charheight;
```

`inquire_charsize()` obtains the *charsize* attribute for the text output primitive.

Inquire Character Spacing

```
inquire_charspace(charspace)
float *charspace;
```

`inquire_charspace()` obtains the *charspace* attribute for the text output primitive.

Inquire Character Up Vector 2

```
inquire_charup_2(dx, dy)
float *dx, *dy;
```

`inquire_charup_2()` obtains the *charup* attribute for the text output primitive.

Inquire Character Up Vector 3

```
inquire_charup_3(dx, dy, dz)
float *dx, *dy, *dz;
```

`inquire_charup_3()` obtains the *charup* attribute for the text output primitive.

Inquire Character Path 2

```
inquire_charpath_2(dx, dy)
float *dx, *dy;
```

`inquire_charpath_2()` obtains the *charpath* attribute for the text output primitive.

Inquire Character Path 3

```
inquire_charpath_3(dx, dy, dz)
float *dx, *dy, *dz;
```

`inquire_charpath_3()` obtains the *charpath* attribute for the text output primitive.

Obtain Justification Attribute

```
inquire_charjust(just)
int *just;
```

`inquire_charjust()` obtains the justification attribute for text strings.

Obtain Current Rasterop (SunCore Extension)

```
inquire_rasterop(rop)
int *rop;    /* XORROP, ORROP, NORMAL */
```

`inquire_rasterop()` determines the current setting of the *rasterop* attribute.

Inquire Character Precision

```
inquire_charprecision(charprecision)
int *charprecision; /* STRING, CHARACTER */
```

`inquire_charprecision()` obtains the *charprecision* attribute for the text output primitive.

Inquire Pick ID

```
inquire_pick_id(pick_id)
int *pick_id;
```

`inquire_pick_id()` obtains the *pick id* attribute for output primitives.

Inquire Marker Symbol

```
inquire_marker_symbol(symbol)
int *symbol;    /* 32 .. 127 */
```

`inquire_marker_symbol()` obtains the current value of the marker symbol.

Obtain All Primitive Attributes

```
inquire_primitive_attributes(attributes)
struct {
    int lineindx, fillindx, textindx;
    int linestyl, polylinestyl, polyedgestyl;
    float linwidth;
    int pen, font;
    float charwidth, charheight;
    float charupx, charupy, charupz, charupw;
    float charpathx, charpathy, charpathz, charpathw;
    float charspacex, charspacey, charspacez, charspacew;
    int chjust, chquality;
    int marker, pickid, rasterop;
} *attributes;
```

`inquire_primitive_attributes()` is a composite function which provides a means to obtain all the primitive attributes in a single function call.

6.4. Retained Segment Static Attributes

There is only one static attribute for segments. This is the *image transformation type* attribute. This attribute can take on one of five values:

NONE — Retained segment on which no translation, scaling, or rotation can be performed.

XLATE2 — Translatable retained segment. The segment can be moved (translated) in 2D (x and y of NDC space).

XFORM2 — Fully transformable retained segment. The segment can be moved (translated), rotated, and scaled (have its size changed) in 2D (x and y of NDC space).

XLATE3 — Translatable retained segment. The segment can be moved (translated) in 3D (x and y of NDC space).

XFORM3 — Fully transformable retained segment. The segment can be moved (translated), rotated, and scaled (have its size changed) in 3D (x, y and z of NDC space).

The *image transformation type* attribute is set when a segment is created and cannot be changed at any time during the life of the segment. The default value of *image transformation type* is NONE.

The functions described below are used to set and inquire about the values of *image transformation type*.

Set Image Transformation Type

```
set_image_transformation_type(type)
int type;   /* NONE, XLATE2, XFORM2, XLATE3, XFORM3 */
```

`set_image_transformation_type()` specifies the *image*

transformation type attribute for subsequently created segments.

Inquire Image Transformation Type

```
inquire_image_transformation_type(type)
int *type;  /* NONE, XLATE2, XFORM2, XLATE3, XFORM3 */
```

`inquire_image_transformation_type()` obtains the current value of the *image transformation type* attribute.

Inquire Segment Image Transformation Type

```
inquire_segment_image_transformation_type(segment_name, type)
int segment_name;   /* Name of segment for inquiry */
int *type;  /* NONE, XLATE2, XFORM2, XLATE3, XFORM3 */
```

`inquire_segment_image_transformation_type()` obtains the *image transformation type* for a specified segment.

6.5. Setting Retained Segment Dynamic Attributes

In addition to the one static attribute described above, there are a number of dynamic attributes which apply to segments. Each retained segment has its own set of dynamic attributes, as listed below.

Visibility
indicates whether the segment should have a visible image. There are only two values of this attribute, namely: TRUE and FALSE.

SunCore sets *visibility* to TRUE at initialization time.

Highlighting
indicates whether the segment's image should be highlighted. In *SunCore*, highlighting is done by briefly blinking the segment. There are only two values of the *highlighting* attribute, namely, TRUE and FALSE.

SunCore sets *highlighted* to FALSE at initialization time.

Detectability
indicates whether the retained segment can be detected by the `await_pick()` input primitive. A value of 0 means that the segment is not pickable. If two segments overlap, the one with the greatest value of *detectability* is the one that gets picked. *SunCore* sets *detectability* to the default value of 0 at initialization time.

Image Transformation
indicates how the image of a retained segment is scaled, rotated, or translated. Image transformations are done in NDC space, that is, after all viewing operations have been performed. Image transformations do not compose and do not cumulate. Whenever any function affecting a segment's image transformation is called, the transformation is reset to reflect only the values specified by the call. The *image transformation* attribute of a segment must be consistent with its *image transformation type* attribute (for instance, if the *image transformation type* is XLATE2, it is an error to attempt to rotate the segment).

SunCore sets the default *image transformation* to the identity transformation (that is, no translation, scaling, or rotation) at initialization.

There are two classes of functions for setting retained segment dynamic attributes. One class sets the default attributes for subsequently created segments; the other sets attributes on a named segment basis.

- There is no retained segment called *segment_name*.
- One or more of the attributes is incorrect.
- The segment's *image transformation type* attribute value is incompatible with the requested function.

Set Visibility

```
set_visibility(visibility)
int visibility; /* TRUE or FALSE */
```

`set_visibility()` specifies the default *visibility* attribute for subsequently created segments. This does not affect the visibility of existing segments or the currently open segment.

Set Highlighting

```
set_highlighting(highlighting)
int highlighting;    /* TRUE or FALSE */
```

`set_highlighting()` specifies the default *highlighting* attribute for subsequently created segments.

Set Detectability

```
set_detectability(detectability)
int detectability;  /* 0 thru 2 to the 31rd power */
```

`set_detectability()` specifies the default *detectability* attribute for subsequently created segments.

Set Image Translate 2

```
set_image_translate_2(tx, ty)
float tx, ty;    /* x and y translation values in NDC */
```

`set_image_translate_2()` sets the default image transformation attribute for subsequently created segments. The default image transformation is set to a 2D translate by *tx* and *ty*.

Set Image Transformation 2

```
set_image_transformation_2(sx, sy, a, tx, ty)
float sx, sy;    /* x and y scale factors */
float a; /* rotation value in radians
    counter-clockwise about z axis */
float tx, ty;    /* x and y translation values in NDC */
```

`set_image_transformation_2()` sets the default image transformation for subsequently created segments. The default transformation is set to a 2D scale by *sx* and *sy*, rotation by *a*, and translation by *tx* and *ty*. The order of transformation is:

1. *Scale* about the origin of NDC space.
2. *Rotate* about the origin of NDC space (about the *z* axis). A positive rotation of $\pi/2$ radians will rotate the *x* axis into the *y* axis.
3. *Translate*.

To scale and rotate about a point *x*, *y*, add *dx* to *tx* and add *dy* to *ty*, where

$$dx = x - (x*sx*\cos(a) - y*sy*\sin(a))$$

$$dx = y - (x*sx*\sin(a) + y*sy*\cos(a))$$

Set Image Translate 3

```
set_image_translate_3(tx, ty, tz)
float tx, ty, tz; /* x, y, and z Translation Values in NDC */
```

`set_image_translate_3()` sets the default image transformation attribute, in NDC space, for subsequently created segments. The default image transformation is set to a 3D translate by *tx*, *ty* and *tz*.

Set Image Transformation 3

```
set_image_transformation_3(sx, sy, sz, ax, ay, az, tx, ty, tz)
float sx, sy, sz;   /* x, y, and z Scale Factors */
float ax, ay, az; /* Rotation Values in radians clockwise */
            /* about the x, y, and z axes */
float tx, ty, tz; /* x, y, and z Translation Values in NDC */
```

`set_image_transformation_3()` sets the default image transformation attribute for subsequently created segments. The default image transformation is set to a 3D scale by *sx*, *sy*, *sz*, a 3D rotation by *ax ay*, *az*, and a 3D translation by *tx*, *ty*, *tz*. The order of transformation is:

1. *Scale* about (0.0, 0.0, 0.0) in NDC space,
2. *Rotate* about (0.0, 0.0, 0.0) in NDC space, first about the *x*-axis, then about the *y*-axis, and then about the *z*-axis. Since NDC space is a left-handed coordinate system, rotations are computed using the left-hand rule. When the origin is viewed from the positive side of the axis of rotation, clockwise rotations correspond to positive rotations.
3. *Translate*.

Set Segment Visibility

```
set_segment_visibility(segment_name, visibility)
int segment_name;
int visibility; /* TRUE or FALSE */
```

`set_segment_visibility()` specifies the *visibility* attribute for the named segment. When *visibility* is set to FALSE, the segment is erased from the view surfaces. The segment is redrawn again when *visibility* is set to TRUE.

Set Segment Highlighting

```
set_segment_highlighting(segment_name, highlighting)
int segment_name;
int highlighting;   /* TRUE or FALSE */
```

`set_segment_highlighting()` specifies the *highlighting* attribute for the named segment. When *highlighting* is set to TRUE, the segment is blinked once.

Set Segment Detectability

```
set_segment_detectability(segment_name, detectability)
int segment_name;
int detectability;  /* 0 thru 2 to the 31rd power */
```

`set_segment_detectability()` specifies the *detectability* attribute for the named segment. When *detectability* is set to 0, the segment cannot be picked by the `await_pick()` input function. If two segments overlap, the segment with the greatest *detectability* is picked.

Set Segment Image Translate 2

```
set_segment_image_translate_2(segment_name, tx, ty)
int segment_name;
float tx;   /* x Translation Value in NDC */
float ty;   /* y Translation Value in NDC */
```

`set_segment_image_translate_2()` sets the image transformation attribute for the named segment. The image transformation is set to a 2D translate by *tx*, *ty*. The named segment is erased from the view surface and then redrawn after the new image transformation is applied. This may be done while the segment is open.

Set Segment Image Transformation 2

```
set_segment_image_transformation_2(segment_name,
    sx, sy, a, tx, ty)
int segment_name;
float sx;   /* x Scale Factor */
float sy;   /* y Scale Factor */
float a; /* Rotation Value in radians
    clockwise about z axis */
float tx;   /* x Translation Value in NDC */
float ty;   /* y Translation Value in NDC */
```

`set_segment_image_transformation_2()` sets the image transformation attribute for the named segment. The image transformation is set to a 2D scale by *sx* and *sy*, a 2D rotation by *a*, and a 2D translation by *tx* and *ty*. The order of transformation is:

1. *Scale* about the origin of NDC space.
2. *Rotate* about the origin of NDC space (about the *z* axis). A positive rotation of $\pi/2$ radians will rotate the x axis into the y axis.
3. *Translate*.

To scale and rotate about a point *x*, *y*, add *dx* to *tx* and add *dy* to *ty*, where

$$dx = x - (x*sx*\cos(a) - y*sy*\sin(a))$$

$$dx = y - (x*sx*\sin(a) + y*sy*\cos(a))$$

The named segment is erased from the view surface and then redrawn after the new image transformation is applied. This may be done while the segment is open.

Set Segment Image Translate 3

```
set_segment_image_translate_3(segment_name, tx, ty, tz)
int segment_name;
float tx;   /* x Translation Value in NDC */
float ty;   /* y Translation Value in NDC */
float tz;   /* z Translation Value in NDC */
```

`set_segment_image_translate_3()` sets the image transformation attribute for the named segment. The image transformation is set to a 3D translate by *tx*, *ty*, *tz*. The named segment is erased from the view surface and then redrawn after the new image transformation is applied. This may be done while the segment is open.

Set Segment Image Transformation 3

```
set_segment_image_transformation_3(segment_name,
    sx, sy, sz, ax, ay, az, tx, ty, tz)
int segment_name;
float sx; /* x Scale Factor */
float sy; /* y Scale Factor */
float sz; /* z Scale Factor */
float ax; /* Rotation Value in radians clockwise
    about the x axis */
float ay; /* Rotation Value in radians clockwise
    about the y axis */
float az; /* Rotation Value in radians clockwise
    about the z axis */
float tx; /* x Translation Value in NDC */
float ty; /* y Translation Value in NDC */
float tz; /* z Translation Value in NDC */
```

`set_segment_image_transformation_3()` sets the image transformation attribute for the named segment. The image transformation is set to a 3D scale by *sx*, *sy*, *sz*, a 3D rotation by *ax*, *ay*, *az*, and a 3D translation by *tx*, *ty*, *tz*. The order of transformation is:

1. *Scale* about (0.0, 0.0, 0.0) in NDC space.
2. *Rotate* about (0.0, 0.0, 0.0) in NDC space, first about the *x*-axis, then about the *y*-axis, and then about the *z*-axis. Since NDC space is a left-handed coordinate system, rotations are computed using the left-hand rule. When the origin is viewed from the positive side of the axis of rotation, clockwise rotations correspond to positive rotations.
3. *Translate*.

The named segment is erased from the view surface and then redrawn after the new image transformation is applied. This may be done while the segment is open.

6.6. Inquiring Retained Segment Dynamic Attributes

The functions described below are for inquiring the settings of the dynamic attributes for retained segments. There are two classes of functions for inquiring retained segment dynamic attributes. One class obtains the default attributes for subsequently created segments and the other obtains attributes on a named segment basis.

- There is no segment called *segment name*.
- The default image transformation attribute value is of a more complex type than the inquiry function used.
- The segment's *image transformation type* attribute value is incompatible with the requested function.
- The segment's *image transformation type* attribute value is of a more complex type than the inquiry function used.

Inquire Visibility

```
inquire_visibility(visibility)
int *visibility;     /* TRUE or FALSE */
```

`inquire_visibility()` obtains the default *visibility* attribute for subsequently created segments.

Inquire Highlighting

```
inquire_highlighting(highlighting)
int *highlighting;  /* TRUE or FALSE */
```

`inquire_highlighting()` obtains the default *highlighting* attribute for the subsequently created segments.

Inquire Detectability

```
inquire_detectability(detectability)
int *detectability; /* 0 thru 2 to the 31rd power */
```

`inquire_detectability()` obtains the default *detectability* attribute for the subsequently created segments.

Inquire Image Translate 2

```
inquire_image_translate_2(tx, ty)
float *tx, *ty; /* x and y Translation Values in NDC */
```

`inquire_image_translate_2()` obtains the 2D translation components of the default image transformation for subsequently created segments.

Inquire Image Transformation 2

```
inquire_image_transformation_2(sx, sy, a, tx, ty)
float *sx, *sy; /* x and y Scale Factors */
float *a; /* Rotation Value in radians
    clockwise about the z axis */
float *tx, *ty; /* x and y Translation Values in NDC */
```

`inquire_image_transformation_2()` obtains the 2D scale factor, rotation, and translation components of the default image transformation attribute for subsequently created segments.

Inquire Image Translate 3

```
inquire_image_translate_3(tx, ty, tz)
float *tx, *ty, *tz; /* x, y, and z Translation Values in NDC *
```

`inquire_image_translate_3()` obtains the 2D translation components of the default image transformation attribute for subsequently created segments.

Inquire Image Transformation 3

```
inquire_image_transformation_3(sx, sy, sz, ax, ay, az, tx, ty, tz)
float *sx, *sy, *sz;    /* x, y, and z Scale Factors */
float *ax, *ay, *az;    /* Rotation Values in radians clockwise
        about the x, y, and z axes */
float *tx, *ty, *tz;    /* x, y, and z Translation Values in NDC *
```

`inquire_image_transformation_3()` obtains the 3D scale factor, rotation, and translation components of the default image transformation attribute for subsequently created segments.

Inquire Segment Visibility

```
inquire_segment_visibility(segment_name, visibility)
int segment_name;
int *visibility;    /* TRUE or FALSE */
```

`inquire_segment_visibility()` obtains the *visibility* attribute for the named segment.

Inquire Segment Highlighting

```
inquire_segment_highlighting(segment_name, highlighting)
int segment_name;
int *highlighting;  /* TRUE or FALSE */
```

`inquire_segment_highlighting()` obtains the *highlighting* attribute for the named segment.

Inquire Segment Detectability

```
inquire_segment_detectability(segment_name, detectability)
int segment_name;
int *detectability; /* 0 thru 2 to the 31rd power */
```

`inquire_segment_detectability()` obtains the *detectability* attribute for the named segment.

Inquire Segment Image Translate 2

```
inquire_segment_image_translate_2(segment_name, tx, ty)
int segment_name;
float *tx;  /* x Translation Value in NDC */
float *ty;  /* y Translation Value in NDC */
```

`inquire_segment_image_translate_2()` obtains the 2D translation components of the named segment's image transformation attribute.

Inquire Segment Image Transformation 2

```
inquire_segment_image_transformation_2(segment_name,
    sx, sy, a, tx, ty)
int segment_name;
float *sx;  /* x Scale Factor */
float *sy;  /* y Scale Factor */
float *a;   /* Rotation Value in radians clockwise
        about the z axis */
float *tx;  /* x Translation Value in NDC */
float *ty;  /* y Translation Value in NDC */
```

`inquire_segment_image_transformation_2()` obtains the 2D scale factor, rotation, and translation components of the named segment's image transformation attribute.

Inquire Segment Image Translate 3

```
inquire_segment_image_translate_3(segment_name, tx, ty, tz)
int segment_name;
float *tx;  /* x Translation Value in NDC */
float *ty;  /* y Translation Value in NDC */
float *tz;  /* z Translation Value in NDC */
```

`inquire_segment_image_translate_3()` obtains the 3D translation components of the named segment's image transformation attribute.

Inquire Segment Image Transformation 3

```
inquire_segment_image_transformation_3(segment_name,
    sx, sy, sz, ax, ay, az, tx, ty, tz)
int segment_name;
float *sx;  /* x Scale Factor */
float *sy;  /* y Scale Factor */
float *sz;  /* z Scale Factor */
float *ax;  /* Rotation Value in radians clockwise
               about the x axis */
float *ay;  /* Rotation Value in radians clockwise
               about the y axis */
float *az;  /* Rotation Value in radians clockwise
               about the z axis */
float *tx;  /* x Translation Value in NDC */
float *ty;  /* y Translation Value in NDC */
float *tz;  /* z Translation Value in NDC */
```

`inquire_segment_image_transformation_3()` obtains the 3D scale factor, rotation, and translation components of the named segment's image transformation attribute.

7

Input Primitives

7

Input Primitives

SunCore supports several *logical input devices* providing for interactive use of the graphics system. The physical input devices provided are the keyboard and the mouse. The mouse is versatile in that it can be used both as a pointer and a button device.

In the terminology of the ACM Core specification, input devices fall into two distinct classes, namely: devices that generate events, and devices that may only be sampled for position or numerical values. *SunCore* supports the ACM Core standard level 2 input (synchronous); hence no event generation or event queue is supported. The supported logical devices in *SunCore* are:

Table 7-1 *Input Devices Supported By SunCore*

Device	*Descruption*
PICK	identifies a segment or a primitive within a segment. *SunCore* uses the mouse as a PICK device.
KEYBOARD	provides alphanumeric information to the application program.
BUTTON	provides a means of choosing among several alternatives. In *SunCore*, the three BUTTON devices are on the mouse.
STROKE	generates a sequence of positions in NDC space. In *SunCore*, the STROKE device is the mouse.
LOCATOR	provides a position in NDC space. *SunCore* uses the mouse as the LOCATOR device.
VALUATOR	provides a scalar value to the application program which samples it. *SunCore* uses the mouse as the valuator device.

A logical input device must be initialized before it can be used.

7.1. Initializing and Terminating Input Devices

The functions described in the sections that follow are used to initialize and terminate input devices. These functions are normally called at the beginning and end of a *SunCore* application program.

Initialize a Specific Device

```
initialize_device(device_class, device_number)
int device_class;   /* PICK, KEYBOARD, STROKE */
            /* LOCATOR, VALUATOR, BUTTON */

int device_number;  /* There are: */
            /* 1 PICK device */
            /* 1 KEYBOARD device */
            /* 1 STROKE device */
            /* 3 BUTTON devices */
            /* 1 LOCATOR device */
            /* 1 VALUATOR device */
```

`initialize_device()` initializes a specific logical device. This function must be called before accessing any of the input devices. An initialized input device which uses position information from the mouse must be associated with an initialized view surface (as an echo surface) before valid data can be read from the device. See Appendix B for details.

Note: that if the KEYBOARD device is initialized and the program crashes before the KEYBOARD device is terminated, the tty will not echo and cbreak will be set. To recover from this condition, type 'reset' followed by a carriage return.

- The device specified by *device_number* is not initialized.
- The device specified by *device_number* is already initialized.

Disable a Specific Device

```
terminate_device(device_class, device_number)
int device_class;   /* PICK, KEYBOARD, STROKE */
            /* LOCATOR, VALUATOR, BUTTON */

int device_number;  /* There are: */
            /* 1 PICK device */
            /* 1 KEYBOARD device */
            /* 1 STROKE device */
            /* 3 BUTTON devices */
            /* 1 LOCATOR device */
            /* 1 VALUATOR device */
```

`terminate_device()` disables a specific device.

- The device specified by *device_number* is not enabled.

7.2. Device Echoing

Device echoing means that *SunCore* can provide a visible indication to the user that the system has seen the input from a specific input device.

SunCore provides the means whereby the application programmer can control the way in which input devices are echoed to the user of the graphics system.

Firstly, the types of echoing for each device are defined here. The tables below describe the types of echoing for specific devices.

Table 7-2 *Echoing for PICK Device*

Echo Type	*Actions Performed*
0	No echo
1	*SunCore* blinks the picked segment briefly. A printer's fist (pointing finger) indicates the position of the PICK device.
2	A printer's fist (pointing finger) indicates the position of the PICK device. *SunCore* does not blink the picked segment.

Table 7-3 *Echoing for KEYBOARD Device*

Echo Type	*Actions Performed*
0	No echo
1	The string which the user typed on the KEYBOARD device is echoed on the screen starting at the echo reference position.

Table 7-4 *Echoing for BUTTON Device*

Echo Type	*Actions Performed*
0	No echo
1	No echo

Table 7-5 *Echoing for STROKE Device*

Echo Type	*Actions Performed*
0	No echo
1	a printers fist (pointing finger) sign is displayed at the cursor position.
2	A string of dots is drawn to follow the path of the cursor. (not implemented)
3	A solid line is drawn to follow the path of the cursor. (not implemented)
4	a printers fist sign is displayed at the final position of the cursor. (not implemented)

Table 7-6 *Echoing for LOCATOR Device*

Echo Type	*Actions Performed*
0	No echo
1	A printers fist (pointing finger) sign is displayed at the position of the LOCATOR device.
2	A solid line is drawn connecting the echo reference point with the LOCATOR.
3	A solid line is drawn connecting the echo reference point with the *x* coordinate of the LOCATOR.
4	A solid line is drawn connecting the echo reference point with the *y* coordinate of the LOCATOR.
5	A solid line is drawn connecting the echo reference point with either the *x* coordinate, or the *y* coordinate, of the LOCATOR, whichever is farthest from the echo reference point.
6	A box is drawn with the position of the LOCATOR as one corner, and the echo reference point as the opposite corner.

Table 7-7 *Echoing for VALUATOR Device*

Echo Type	*Actions Performed*
0	No echo
1	The current value of the valuator is displayed on the screen starting at the echo reference point.
2 – 11	*SunCore* does not perform the actions as described in the ACM Core specification, which sets the values of the valuator into various parameters of the *image_transformation_type* attribute of retained segments. *SunCore* leaves this up to the application program.

Define Type of Echo for Device

```
set_echo(device_class, device_number, echo_type)
int device_class;    /* PICK, KEYBOARD, STROKE, */
                     /* LOCATOR, VALUATOR, BUTTON */
int device_number;
int echo_type;
```

`set_echo()` determines the echo type for a input device.

Define Type of Echo for a Group of Devices

```
set_echo_group(device_class, device_number_array, n, echo_type)
int device_class;   /* PICK, KEYBOARD, STROKE, */
            /* LOCATOR, VALUATOR, BUTTON */
int device_number_array[];
int n;  /* number of devices in array */
int echo_type;
```

`set_echo_group()` determines the echo type for an input device class.

Define Echo Reference Point

```
set_echo_position(device_class, device_number, echo_x, echo_y)
int device_class;   /* PICK, KEYBOARD, STROKE, */
            /* LOCATOR, VALUATOR, BUTTON */
int device_number;
float echo_x;       /* x Coordinate of Echo Point */
float echo_y;       /* y Coordinate of Echo Point */
```

`set_echo_position()` specifies the position, in NDC space, which will be used as the echo reference point. The coordinates must lie within the bounds of NDC space, or `set_echo_position()` will set the echo reference point to be the point in NDC space closest to the specified point. The echo reference point that this function defines is used for certain types of echo such as rubber band LOCATOR echo.

Define View Surface for Echo

```
set_echo_surface(device_class, device_number, surface_name)
int device_class;   /* PICK, KEYBOARD, STROKE, */
            /* LOCATOR, VALUATOR, BUTTON */
int device_number;
struct vwsurf *surface_name;    /* See Appendix B */
```

`set_echo_surface()` specifies the viewing surface on which echoing will be done. An initialized input device which uses position information from the mouse must be associated with an initialized view surface (as an echo surface) before valid data can be read from the device. See Appendix B for details. If a NULL pointer is given for the *surface_name* argument, any association of the specified input device with an echo surface is ended.

7.3. Setting Input Device Parameters

The functions described in the sections that follow are used to define certain parameters for each of the logical input devices. These functions are normally called at the beginning of a *SunCore* application program.

Initialize LOCATOR Position

```
set_locator_2(locator_number, x, y)
int locator_number;
float x;
float y;
```

`set_locator_2()` sets the initial LOCATOR position in NDC space.

Initialize Value and Range for VALUATOR Device

```
set_valuator(valuator_number, initial_value, low, high)
int valuator_number;
float initial_value;
float low;
float high;
```

`set_valuator()` sets the value and range for the valuator device. The default values are: *initial_value=0.0*, *low=0.0*, and *high=1.0*.

Initialize KEYBOARD Parameters

```
set_keyboard(keyboard_number, buffer_size,
    initial_string, initial_cursor_position)
int keyboard_number;
int buffer_size;
char *initial_string;
int initial_cursor_position;
```

`set_keyboard()` sets the size of the character buffer for the KEYBOARD device, the initial character string, and the initial character cursor counting from the echo reference position. *SunCore* uses default values of *buffer_size*=80, *initial_string*="enter:", and *initial_cursor_position*=7. The maximum buffer_size and the maximum length of initial_string are 80 characters.

Initialize STROKE Device

```
set_stroke(stroke_number, buffer_size, distance, time)
int stroke_number;  /* Device Number */
int buffer_size;    /* not used */
float distance;     /* Minimum distance to move */
int time;       /* not used */
```

`set_stroke()` sets parameters for the STROKE device. The *buffer_size* argument is the maximum number of *x*, *y* points in a STROKE. The *distance* argument is the minimum distance, in NDC space, which the mouse must move before a new point is added to the *x*, *y* list comprising the STROKE. The default setting is *distance=0.01*.

Initialize PICK Device

```
set_pick(pick-number, aperture)
int pick-number;    /* device number */
float aperture;     /* device aperture */
```

`set_pick()` sets the aperture for the PICK device. The *aperture* argument provides control over the 'sensitivity' of the PICK device. A square is defined with its center at the cursor position and with sides of length 2 * *aperture*. Segments that intersect this square can be picked. *aperture* is given in NDC space. An error is returned if the *pick-number* is incorrect or if the *aperture* ≤0.0. The default aperture square has two pixels per side.

7.4. Reading From Input Devices

SunCore has several functions for interrogating input devices. These function allow the application programmer a great deal of flexibility in user-interface design.

Wait for BUTTON Device

```
await_any_button(time, button_number)
int time;    /* Time in microseconds to wait */
int *button_number; /* BUTTON which was hit */
```

`await_any_button()` waits for the user to click any of the BUTTON devices. `await_any_button()` waits for the user to click any BUTTON device, or until the time specified by the *time* parameter expires. If the *time* argument is exactly zero, the BUTTON devices are checked once, then the function returns to the caller immediately.

If a BUTTON device is clicked before *time* expires, the number of the BUTTON device is returned in the *button_number* parameter. If the user does not click any BUTTON device before *time* expires, the function returns a BUTTON device number of zero.

For the mouse, BUTTON device numbers 1, 2, and 3 represent the left, middle, and right buttons, respectively, when the buttons are facing *away* from the user.

Wait for PICK Device

```
await_pick(time, pick_number, segment_name, pick_id)
int time;    /* Time in microseconds to wait */
int pick_number;
int *segment_name;
int *pick_id;
```

`await_pick()` waits for the user to pick an output primitive within a visible and detectable retained segment. `await_pick()` waits for the user to click the left hand button on the mouse, or until the time specified by the *time* parameter expires. If the *time* argument is exactly zero, the function tests the button once, and if the button has been clicked, performs the pick operation.

If the button is clicked before *time* expires, the function returns the *segment_name* of the segment that the PICK device is pointing at, and the *pick_id* parameter is set to the value of the *pick_id* attribute of the primitive that was picked. If the user does not click any mouse button before *time* expires, or no segment is found where the user points, the function sets the *segment_name* and *pick_id* parameters to zero.

`await_pick()` only searches those segments which are visible and detectable and appear on the echo surface of the specified PICK device. Primitives within a segment have bounded volume descriptors. The square pick aperture must intersect one of these 'extents' in order that the *segment_name* and *pick_id* be returned. If more than one segment is at the point, the segment with the highest value of the detectability attribute is returned. Detectability may be set to zero to prevent a segment from being picked.

- The specified PICK device does not exist.

Wait for Input from the KEYBOARD

```
await_keyboard(time, keyboard_number, input_string, length)
int time;   /* Time in microseconds to wait */
int keyboard_number;
char *input_string;
int *length;
```

`await_keyboard()` waits for the user to type a line of input on the KEYBOARD device. `await_keyboard()` waits for the user to enter data at the KEYBOARD device, or until the time specified by the *time* parameter expires. If the *time* argument is exactly zero, the function tests once to see if a character has been typed, and then returns to the caller.

If any data is entered at the KEYBOARD device before *time* expires, the function returns the typed characters in an array pointed to by *input_string*. The length of this character string is returned in *length*. The string is null terminated. If the user does not enter any data before *time* expires, the function sets the *length* parameter to zero. If a carriage-return or newline character is typed, the function returns with the input string containing a newline character as the last non-null character.

- The specified KEYBOARD device does not exist.

Wait for User to Draw a Curve

```
await_stroke_2(time, stroke_number, array_size,
    x_array, y_array, number_points)
int time;   /* Time in microseconds to wait */
int stroke_number;  /* STROKE device to wait for */
int array_size; /* Maximum size of x and y arrays */
float x_array[];
float y_array[];
int *number_points; /* Number of x, y coordinates
    actually read */
```

`await_stroke_2()` waits for the user to draw a curve, consisting of a list of points in NDC space, using the mouse. A curve in this context means a string of line segments. `await_stroke_2()` waits for the user to draw a curve using the mouse, or until the time specified by the *time* parameter expires. If the *time* argument is exactly zero, the function tests once to see if a curve has been drawn, and then returns to the caller.

The curve starts at the current position of the LOCATOR, and finishes when the user clicks button 3 on the mouse. When the function returns, the number of *x, y* coordinates actually read is returned in the *number_points* argument. When the number of points read equals *array_size* the function returns before time expires. Note: The BUTTON device must be initialized for `await_stroke_2()` to work.

Read LOCATOR When BUTTON Clicked

```
await_any_button_get_locator_2(time, locator_number,
    button_number, x, y)
int time;   /* Time in microseconds to wait */
int locator_number; /* LOCATOR device to wait for */
int *button_number; /* BUTTON which was clicked */
float *x, *y;   /* Returned point in NDC */
```

`await_any_button_get_locator_2()` waits for the user to click any of the mouse buttons. When the button is clicked, the function returns the current NDC coordinates of the LOCATOR.
`await_any_button_get_locator_2()` waits for the user to click any mouse button, or until the time specified by the *time* argument expires. If the *time* argument is exactly zero, the function checks if any buttons have been clicked immediately and then returns.

If the time expires before the user has clicked any of the mouse buttons, the function returns a zero in the *button_number* argument.

Read VALUATOR When BUTTON Clicked

```
await_any_button_get_valuator(time, valuator_number,
    button_number, value)
int time;   /* Time in microseconds to wait */
int valuator_number;    /* VALUATOR number to read from */
int *button_number; /* BUTTON which was clicked */
float *value;   /* Value of valuator */
```

`await_any_button_get_valuator()` waits for the user to click any of the mouse buttons, or for a specified time. When the button is clicked, the function returns the current value of the valuator.
`await_any_button_get_valuator()` waits for the user to click any mouse button, or until the time specified by the *time* argument expires. If the *time* argument is exactly zero, the function checks if any buttons have been clicked and then returns immediately.

If the user clicks one of the mouse buttons, the function returns with the value of the valuator, and the number of the button which was clicked. If the time expires before the user has clicked any of the mouse buttons, the function returns a zero in the *button_number* argument. Movement of the mouse left or right lowers or raises the value of the valuator. Note: The BUTTON device must be initialized for `await_any_button_get_valuator()` to work.

Low Level Mouse Support (SunCore extension)

```
get_mouse_state(device_class, device_number, x, y, buttons)
int device_class;   /* PICK, STROKE, */
            /* LOCATOR, VALUATOR, BUTTON */
int device_number;
float *x, *y;
int *buttons;
```

`get_mouse_state()` reads the low level mouse *x*, *y* and button information corresponding to a particular input device. The buttons are up-down encoded, and the location of the mouse is in NDC space.

Bit 0 of *buttons* is the right-hand mouse button.

Bit 1 of *buttons* is the middle mouse button.

Bit 2 of *buttons* is the left-hand mouse button.

A zero bit means that the button is *up*, while a one bit means that the button is *down*.

7.5. Inquiring Input Status Parameters

The functions described in the sections that follow are used to inquire various parameters of the logical input devices.

Obtain Type of Echo for Device

```
inquire_echo(device_class, device_number, echo_type)
int device_class;   /* PICK, KEYBOARD, STROKE, */
            /* LOCATOR, VALUATOR, BUTTON */
int device_number;
int *echo_type;
```

`inquire_echo()` obtains the echo_type for the specified device.

Obtain Echo Reference Point

```
inquire_echo_position(device_class, device_number,
    echo_x, echo_y)
int device_class;
int device_number;
float *echo_x;      /* x Coordinate of Echo Point */
float *echo_y;      /* y Coordinate of Echo Point */
```

`inquire_echo_position()` obtains the position, in NDC space, of the echo reference point for the specified device.

Obtain View Surface for Echo

```
inquire_echo_surface(device_class, device_number, surface_name)
int device_class;
int device_number;
struct vwsurf *surface_name;
```

`inquire_echo_surface()` obtains the viewing surface on which echoing is done for the specified device.

Obtain Initial LOCATOR Position

```
inquire_locator_2(locator_number, x, y)
int locator_number;
float *x;
float *y;
```

`inquire_locator_2()` obtains the initial position of the specified LOCATOR in NDC space.

Obtain Value and Range for VALUATOR Device

```
inquire_valuator(valuator_number, initial_value, low, high)
int valuator_number;
float *initial_value;
float *low;
float *high;
```

`inquire_valuator()` obtains the value and range for the specified valuator device.

Obtain KEYBOARD Parameters

```
inquire_keyboard(keyboard_number, buffer_size, initial_string,
    initial_cursor_position)
int keyboard_number;
int *buffer_size;
char *initial_string;
int *initial_cursor_position;
```

`inquire_keyboard()` obtains the size of the character buffer, the initial character string, and the initial character cursor for the specified KEYBOARD device.

Obtain STROKE Device Parameters

```
inquire_stroke(stroke_number, buffer_size, distance, time)
int stroke_number;  /* device number */
int *buffer_size;   /* not used */
float *distance;    /* minimum distance to move in NDC */
int *time;      /* not used */
```

`inquire_stroke()` obtains the buffer size, distance, and time parameters for the specified STROKE device.

A

Deviations from ACM SIGGRAPH Core

A

Deviations from ACM SIGGRAPH Core

This appendix points out specific differences between the *SunCore* graphics package and the ACM SIGGRAPH Core Specification. In addition to differences noted here, *SunCore* has numerous extensions to the ACM Core which are documented in the main body of this manual.

A.1. Unimplemented Functions

Here is a list of those functions which *SunCore* does not implement:

Table A-1 *Unimplemented Primitive Attribute Functions*

Primitive Attribute Functions
`set_charjust`
`inquire_charjust`

Table A-2 *Unimplemented Synchronous Input Functions*

Synchronous Input Functions	
`await_stroke_3`	`inquire_pick`
`initialize_group`	`inquire_stroke_dimension`
`inquire_button`	`set_all_buttons`
`inquire_echo_segments`	`set_button`
`inquire_input_capabilities`	`set_echo_segment`
`inquire_input_device_characteristics`	`set_locator_3`
`inquire_locator_3`	`set_locport_2`
`inquire_locator_dimension`	`set_locport_3`
`inquire_locport_2`	`terminate_group`
`inquire_locport_3`	

Table A-3 *Unimplemented Asynchronous Input Functions*

Asynchronous Input Functions	
enable_device	enable_group
disable_device	disable_group
disable_all	read_locator_2
read_locator_3	read_valuator
await_event	flush_device_events
flush_group_events	flush_all_events
associate	disassociate
disassociate_device	disassociate_group
disassociate_all	get_pick_data
get_keyboard_data	get_stroke_data_2
get_stroke_data_3	get_locator_data_2
get_locator_data_3	get_valuator_data
inquire_device_associations	inquire_device_status

Table A-4 *Unimplemented Control Functions*

Control Functions	
inquire_output_capabilities	inquire_selected_surfaces
set_immediate_visibility	make_picture_current
inquire_control_status	set_visibilities
log_error	

Table A-5 *Unimplemented Escape Functions*

Escape Functions
escape
inquire_escape

A.2. Other Differences

The sections that follow describe other differences between the Core specification and *SunCore*.

Text

SunCore does not have the charplane primitive attribute; instead, the charpath, charup, and charspace attributes are used to specify text orientation as described in the manual. The current release of *SunCore* has no STROKE precision text and no text justification. The `inquire_text_extent_2()` and `inquire_text_extent_3()` functions do not take a view surface name as an argument. The text inquiry functions only return meaningful values when the current *charprecision* attribute is CHARACTER.

Raster Extensions

SunCore contains several of the proposed raster extensions to the ACM Core and other raster functions. Thus there are no color or intensity primitive attributes. Instead a color lookup table model is used. There are several primitive attributes which are indices into lookup tables. In addition, hidden surfaces are supported on color view surfaces. This requires a second parameter to the

`initialize_view_surface()` function.

Miscellaneous

SunCore adds these functions:

Table A-6 *SunCore Extensions*

SunCore Extension Functions
`set_image_translate_3`
`inquire_image_translate_3`
`set_segment_image_translate_3`
`inquire_segment_image_translate_3`

Table A-7 *SunCore Replacements*

Core Function	*SunCore Replacement*
`set_primitive_attributes_2` `set_primitive_attributes_3`	`set_primitive_attributes`
`inquire_primitive_attributes_2` `inquire_primitive_attributes_3`	`inquire_primitive_attributes`

Default values for many *SunCore* system parameters differ from those of the ACM Core.

There are restrictions on `set_world_coordinate_matrix_2()` and `set_world_coordinate_matrix_3()` as described in the manual.

As described in the manual, some of the echo types for input functions in the ACM Core are not implemented.

The marker symbol primitive attribute deviates from the ACM Core as described in the manual.

Batching of updates only applies to dynamic segment attributes as described in the manual.

View surfaces initialized for hidden-surface elimination do not support dynamic segment attributes of highlighting, transformation, or translation. `initialize_view_surface()` can optionally suppress clearing the view surface when it is initialized.

B

SunCore View Surfaces

B

SunCore View Surfaces

SunCore supports several types of view surfaces and multiple simultaneous instances of any type, subject to the hardware resources of the workstation on which a *SunCore* program is being run. The current release allows up to five view surfaces to be active at any time. This appendix gives implementation details of *SunCore* view surfaces and provides information on initializing them.

B.1. The `vwsurf` Structure

View surface names in *SunCore* are structures. The following declaration and definitions are contained in the header file `<usercore.h>`:

```
#define DEVNAMESIZE 20

struct vwsurf {
    char screenname[DEVNAMESIZE];
    char windowname[DEVNAMESIZE];
    int windowfd;
    int (*dd)();
    int instance;
    int cmapsize;
    char cmapname[DEVNAMESIZE];
    int flags;
    char **ptr;
    };

#define NULL_VWSURF {"", "", 0, 0, 0, 0, "", 0, 0}
#define DEFAULT_VWSURF(ddname) \
    {"", "", 0, ddname, 0, 0, "", 0, 0}
#define VWSURF_NEWFLG   1
```

After initialization via the function `initialize_view_surface()`, a `vwsurf` structure represents a specific instantiation of a particular type of view surface. The elements of the `vwsurf` structure completely characterize that instantiation and/or provide information used to initialize the view surface. This appendix refers to members of the `vwsurf` structure using the standard C notation, as if the declaration

```
struct vwsurf vwsurf;
```

had been given.

vwsurf.screenname
is a character string which is the name of the physical device on which the

view surface appears (for example, */dev/cgone0*).

vwsurf.windowname
is a character string which is the name of a window device which has been opened for display of the output primitives directed to the view surface (for example, */dev/win10*).

vwsurf.windowfd
is the file descriptor corresponding to this device. Since, for all current *SunCore* view surface types, output display and input device echoing are accomplished through window system functions, these members of the structure are valid even for raw output devices.

vwsurf.dd
is the name of the device-independent/device-dependent interface function through which graphics output to the view surface will pass. This function defines the view surface type. The current *SunCore* view surface types are described below.

vwsurf.instance
identifies the instantiation of a view surface type. It should be set to 0 prior to calling `initialize_view_surface()`. *SunCore* will set this value appropriately if the initialization is successful.

vwsurf.cmapsize
defines the size of the color lookup table for the view surface, and the character string *vwsurf.cmapname* gives its name, which can be used to share a color map between two or more view surfaces on the same physical device. These elements of the vwsurf structure are used only for view surfaces on color devices. Their use is described more fully below.

vwsurf.flags
is a field of one-bit flags. Currently, only one flag, VWSURF_NEWFLG, is defined; this flag is described below.

vwsurf.ptr
is a pointer to an array of character pointers. The array should be terminated by a null pointer. The strings pointed to by the array contain optional information which may be used to initialize the view surface. Details are provided below.

B.2. View Surface Types

A view surface type in *SunCore* is the name of the driver function for the device-independent/device-dependent interface. The name of the function corresponding to the desired view surface type should be put into *vwsurf.dd* prior to calling `initialize_view_surface()` (see the programming examples in Chapters 1 and 8).

The current release of *SunCore* has eight view surface types:

bw1dd
The Sun-1 monochrome bitmap display used as a raw device.

bw2dd
The Sun-2 or Sun-3 monochrome bitmap display used as a raw device.

cg1dd
The Sun-1 color graphics display used as a raw device.

cg2dd
The Sun-2 or Sun-3 color graphics display used as a raw device.

cg4dd
The Sun-3/110 color display used as a raw device.

pixwindd
A monochrome (one bit deep) graphics window within the Suntools window environment. This window may appear on either a color or monochrome display.

cgpixwindd
A color graphics window within the Suntools window environment. This window must appear on a color display.

gp1dd
A Sun-2/160 or Sun-3/160 graphics display with a Graphics Processor option.

gp1pixwindd
A color graphics window within the Suntools window environment running on a Sun-2/160 or Sun-3/160 color graphics display with a Graphics Processor option.

Only view surface types *cg1dd, cg2dd, cg4dd, cgpixwindd, gp1dd,* and *gp1pixwindd* support hidden surface removal. In the discussion above, gray scale devices are considered to be color devices.

The term 'raw device' above implies that the physical device specified by *vwsurf.screenname* is used completely and only for display of graphics output directed to one view surface. This allows somewhat more efficient display of output primitives. It also implies that the user has not started up a Suntools window environment using the device as a desktop.

Low-level device-dependent functions are not part of *SunCore*. For efficiency, such functions are necessary for some applications. The *Pixrect Reference Manual* contains information on low-level functions corresponding to `bw1dd`, `bw2dd`, `cg1dd`, `cg2dd` `cg4dd` and `gp1dd`, (the 'pixrect' level) and `pixwindd`, `cgpixwindd` and `gp1pixwindd` (the 'pixwin' level).

B.3. Choosing a View Surface Type within an Application Program

It may be desirable to write application programs which use different view surface types depending on the environment. The next two subsections provide examples of ways to do this. The next subsection illustrates using a Shell variable, and the subsection after that uses the `get_view_surface()` function to do the job in a more general way. The source for `get_view_surface()` is contained in `/usr/src/sun/suntool/get_view_surface.c`.

Using Shell Variables to Determine the Environment

Examining a Shell environment variable is one way to determine which environment a program is running in. The following example illustrates using either a *bw2dd* (raw Sun-2 or Sun-3 monochrome display) or a *pixwindd* (monochrome window) view surface depending on whether the user is currently in the Suntools window environment. The WINDOW_ME environment variable is normally defined in the user's environment if and only if the window system is being used.

Figure B-1 *Selecting a View Surface from an Environment Variable*

```
int bw2dd();
struct vwsurf rawsurface = DEFAULT_VWSURF(bw2dd);
int pixwindd();
struct vwsurf windowsurface = DEFAULT_VWSURF(pixwindd);

main()
{
    struct vwsurf *surface, *get_surface();

    .
    .
    .
    surface = get_surface();
    initialize_view_surface(surface, FALSE);
    select_view_surface(surface);
    .
    .
    .
}

/* returns a pointer to an appropriate view surface */
struct vwsurf *get_surface()
{
    if (getenv("WINDOW_ME"))
        return(&windowsurface);
    else
        return(&rawsurface);
}
```

The `get_view_surface` Function

The *SunCore* library includes the `get_view_surface()` function which a programmer can use to set up a view surface structure using information from command-line arguments and the environment. A complete listing of `get_view_surface()` appears at the end of this section. `get_view_surface()` has the following declarations for C, FORTRAN, and Pascal:

Table B-1 *Declarations of* `get_view_surface` *in C, FORTRAN, and Pascal*

Language	*Declaration*

C	`get_view_surface(vsptr, argv)`
	`struct vwsurf *vsptr;`
	`char **argv;`

FORTRAN	`getviewsurface(vwsurf)` `integer vwsurf(VWSURFSIZE)`
Pascal	`getviewsurface(var surfacename:` `vwsurf): integer; external;`

The elements of *argv* are pointers to null-terminated strings which are extracted from the command line that started the application program. The following fragment of C code illustrates the use of `get_view_surface()` for C programs:

Figure B-2 `get_view_surface` *Example*

```
main(argc, argv)
int argc;
char **argv;
{
    struct vwsurf vwsurf;

    .
    .
    .
code
    .
    .
    .
    if (get_view_surface(&vwsurf, argv))
        exit(1);
    initialize_view_surface(&vwsurf, FALSE))
    .
    .
    .
more code
    .
    .
    .
}
```

`get_view_surface()` returns zero (0) if it succeeds and non-zero otherwise. The `vwsurf` structure will have *vwsurf.dd* and possibly *vwsurf.screenname* set to appropriate values. Other elements of the structure will be null — the programmer may modify them to suit the application, but it is not necessary.

The only command-line option that `get_view_surface()` currently recognizes is the display_device–d.I option, where *display_device* is the name of the physical display device (*/dev/fb* or */dev/cgone0* for example). The `vwsurf` structure will be set up to run on this device. `get_view_surface()` also

determines if the window system is running on the device, and chooses *vwsurf.dd* appropriately.

Using `get_view_surface()` has a disadvantage in that since it refers to all six *SunCore* types of view surfaces, any program using it will get the code for all six device-independent/device-dependent driver functions linked in. For this reason, the code for `get_view_surface()` is included here. *SunCore* programmers may wish to tailor a version of this code for particular machine configurations and applications in order to make smaller final object code.

The code of `get_view_surface()` contains calls on several functions from `libsunwindow.a` — the *SunView* library. Details of these functions can be found in the *SunView Programmer's Guide* and *SunView System Programmer's Guide.*

Figure B-3 `get_view_surface.c` *Module*

```
/*
    get_view_surface  --  Determines from command-line arguments and
                  the environment a reasonable view surface
                  for a SunCore program to run on.
*/

#include <sunwindow/window_hs.h>
#include <sys/file.h>
#include <sys/ioctl.h>
#include <sun/fbio.h>
#include <stdio.h>
#include <usercore.h>

int bw1dd();        /* All device-independent/device-dependent   */
int bw2dd();        /* routines are referenced in this function.      */
int cg1dd();        /* This means the linker will pull in all of them */
int cg2dd();
int gp1dd();
int pixwindd();
int cgpixwindd();
int gp1pixwindd();

static struct vwsurf nullvs = NULL_VWSURF;

static char *devchk;
static int devhaswindows;

int get_view_surface(vsptr, argv)
struct vwsurf *vsptr;
char **argv;
    {
    int devfnd, fd, chkdevhaswindows();
    char *wptr, dev[DEVNAMESIZE], *getenv();
    struct screen screen;
    struct fbtype fbtype;
```

```
    *vsptr = nullvs;
    devfnd = FALSE;
    if (argv)
        /*
        If command-line arguments are passed, process them using
        win_initscreenfromargv (see the Programmer's Reference Manual
        for the Sun Window System).  The only option used by
        get_view_surface is the -d option, allowing the user to
        specify the display device on which to run.
        */
        {
        win_initscreenfromargv(&screen, argv);
        if (screen.scr_fbname[0] != ' ')
            {
            /* -d option was found */
            devfnd = TRUE;
            strncpy(dev, screen.scr_fbname, DEVNAMESIZE);
            /*
            Check to see if this device has a window system
            running on it.  If so devhaswindows will be TRUE
            following the call to win_enumall.  win_enumall is
            a function in libsunwindow.a.  It takes a function
            as its argument, and applies this function to every
            window being displayed on any screen by the window
            system.  To do this it opens each window and passes
            the windowfd to the function.  The enumeration
            continues until all windows have been tried or the
            function returns TRUE.
            */
            devchk = dev;
            devhaswindows = FALSE;
            win_enumall(chkdevhaswindows);
            }
        }
    if (!devfnd)
        /* No -d option was specified */
        if (wptr = getenv("WINDOW_ME"))
            {
            /*
            Running in the window system.  Find the device from
            which this program was started.
            */
            devhaswindows = TRUE;
            if ((fd = open(wptr, O_RDWR, 0)) < 0)
                {
                fprintf(stderr, "get_view_surface: Can't open %s\n",
                wptr);
                return(1);
                }
            win_screenget(fd, &screen);
            close(fd);
            strncpy(dev, screen.scr_fbname, DEVNAMESIZE);
            }
```

```
        else
            {
            /*
            Not running in the window system.  Assume device is
            /dev/fb.
            */
            devhaswindows = FALSE;
            strncpy(dev, "/dev/fb", DEVNAMESIZE);
            }
    /* Now have device name.  Find device type. */
    if ((fd = open(dev, O_RDWR, 0)) < 0)
        {
        fprintf(stderr, "get_view_surface: Can't open %s\n", dev);
        return(1);
        }
    if (ioctl(fd, FBIOGTYPE, &fbtype) == -1)
        {
        fprintf(stderr, "get_view_surface: ioctl FBIOGTYPE failed on %s\n",
            dev);
        close(fd);
        return(1);
        }
    close(fd);
    /* Now have device type and know if window system is running on it. */
    if (devhaswindows)
        switch(fbtype.fb_type)
            {
        case FBTYPE_SUN1BW:
        case FBTYPE_SUN2BW:
            vsptr->dd = pixwindd;
            break;
        case FBTYPE_SUN1COLOR:
        case FBTYPE_SUN2COLOR:
            vsptr->dd = cgpixwindd;
            break;
        case FBTYPE_SUN2GP:
            vsptr->dd = gp1pixwindd;
            break;
        default:
            fprintf(stderr,
            "get_view_surface: %s is unknown fbtype\n", dev);
            return(1);
            }
    else
        switch(fbtype.fb_type)
            {
        case FBTYPE_SUN1BW:
            vsptr->dd = bw1dd;
            break;
        case FBTYPE_SUN2BW:
            vsptr->dd = bw2dd;
            break;
        case FBTYPE_SUN1COLOR:
```

```
            vsptr->dd = cg1dd;
            break;
        case FBTYPE_SUN2COLOR:
            vsptr->dd = cg2dd;
            break;
        case FBTYPE_SUN2GP:
            vsptr->dd = gp1dd;
            break;
        default:
            fprintf(stderr,
            "get_view_surface: %s is unknown fbtype\n", dev);
            return(1);
            }
    /* Now SunCore device driver pointer is set up. */
    if (!devhaswindows || devfnd)
        /*
        If no window system on device or -d option was specified,
        tell SunCore which device.  Otherwise, let SunCore figure
        out the device itself from WINDOW_GFX so the default
        window will be used if desired.
        */
        strncpy(vsptr->screenname, dev, DEVNAMESIZE);
    return(0);
    }

static int chkdevhaswindows(windowfd)
int windowfd;
    {
    struct screen windowscreen;

    win_screenget(windowfd, &windowscreen);
    if (strcmp(devchk, windowscreen.scr_fbname) == 0)
        {
        /*
        If this window is on the display device we are checking, set
        the flag TRUE.  Return TRUE to terminate the enumeration.
        */
        devhaswindows = TRUE;
        return(TRUE);
        }
    return(FALSE);
    }
```

B.4. Specifying a View Surface for Initialization

It is not necessary to specify every member of the `vwsurf` structure in order to initialize the view surface. If only *vwsurf.dd* is specified, *SunCore* will try to obtain a view surface of the specified type according to a default sequence. A statically allocated `vwsurf` structure may be set up to use this default by initializing the structure via the DEFAULT_VWSURF macro defined in `<usercore.h>`. This is a compile-time initialization. The user may exercise finer control over view surfaces by setting other elements of the structure as described below. Any members which are not specified by the user should be set

to zero (the integer 0, the NULL pointer, or an empty string, as appropriate).

View Surface Specification for Raw Devices

The default action for obtaining a new view surface of a raw device type is to try to open a sequence of devices until one is found which is of the right type and is not already being used. The sequence always starts with */dev/fb*. Then the following names are tried depending on the view surface type:

```
bw1dd - "/dev/bwone0", "/dev/bwone1", ..., "/dev/bwone9"
bw2dd - "/dev/bwtwo0", "/dev/bwtwo1", ..., "/dev/bwtwo9"
cg1dd - "/dev/cgone0", "/dev/cgone1", ..., "/dev/cgone9"
cg2dd - "/dev/cgtwo0", "/dev/cgtwo1", ..., "/dev/cgtwo9"
cg4dd - "/dev/cgfour0", "/dev/cgfour1", ..., "/dev/cgfour9"
gp1dd - "/dev/gpone0a", "/dev/gpone0b", ..., "/dev/gpone3d"
```

If none of the names in the sequence can be successfully opened and verified to be of the correct type and not already in use,
`initialize_view_surface()` fails.

If the user wishes to specify a particular physical device for a view surface, he may set *vwsurf.screenname* to be the device name of that device. The same steps will be taken to try to open the device as for each name in the default sequence. However, if these steps fail, no other names will be tried, and the initialization will fail.

vwsurf.cmapname and *vwsurf.cmapsize* are only used for color view surfaces. For `cg1dd`, `cg2dd`, `cg4dd` and `gp1dd` *vwsurf.cmapsize* is set to 256. If *vwsurf.cmapname* is specified, this name is used as the name of the color map; otherwise *SunCore* will provide a unique name.

No flags are currently defined for use with raw devices.

vwsurf.ptr provides a mechanism for passing optional initialization data to *SunCore*. In the case of raw devices, one such option is currently available — the passing of information about the adjacencies of physical screens. When the user creates a Suntools window environment on a screen, he is also responsible for specifying the relationship of that screen to other screens also running Suntools for purposes of tracking the mouse across multiple screens. The *adjacentscreens* command may be used to do this (see the *SunOS Reference Manual*). However, when a *SunCore* program initializes a new view surface on a raw screen, the user will not previously have been able to inform the system of this adjacency because the new screen was previously not in use. *vwsurf.ptr* may be used to pass adjacency information for the new screen.

If *vwsurf.ptr* is not NULL, it should point to an array of character pointers. Only the first pointer in this array will be used. It should point to a string which is the pathname of a file containing information about the adjacencies of physical display devices. When the user sets up his display devices on his desk he may create a file describing the layout of these devices. For example, the following lines describe a system with two screens, the console frame buffer on the left (which might be a monochrome bitmap display) and a Sun color graphics display on the right:

```
/dev/fb
R: /dev/cgone0
/dev/cgone0
L: /dev/fb
```

By convention, */dev/fb* is the console frame buffer and */dev/cgone0* is the first Sun color graphics display on a system. For each display device in the system, there should be one line giving its name, followed by several lines giving the directions and names of all adjacent screens. Thus all four lines above are necessary, not just the first two. Directions may be indicated as R, L, T, and B for right, left, top, and bottom, or as N, S, E, and W for north, south, east, and west.

View Surface Specification for Window Devices

The default action for obtaining a new view surface of type *pixwindd*, *cgpixwindd* or *gp1pixwindd* is to first test whether the window referred to by the Shell environment variable WINDOW_GFX is already in use as a view surface. If not, a blanket window is inserted over the WINDOW_GFX window and this blanket window becomes the view surface. If WINDOW_GFX has already been used in this manner, the program `/usr/lib/view_surface` is invoked to create a new window on the same physical display device as WINDOW_GFX. This new window becomes the view surface. Thus, if a *SunCore* program is run from the tty subwindow of a Graphics Tool, the first default view surface will occupy the display space covered by the graphics subwindow of the tool. Subsequent default view surfaces will appear as graphics windows, each within a separate *View Surface Tool* on the same screen as the *Graphics Tool*.

This default action may be circumvented in two ways. If *vwsurf.flags* has the VWSURF_NEWFLG set, no attempt is made to take over WINDOW_GFX. A new window within a *View Surface Tool* is opened on the same screen as WINDOW_GFX. If *vwsurf.screenname* is non-empty, a new window within a *View Surface Tool* is opened on the screen specified by *vwsurf.screenname*, provided this device exists and has a Suntools window environment running on it.

For view surfaces of type `cgpixwindd` or `gp1pixwindd`, *vwsurf.cmapsize* and *vwsurf.cmapname* provide a means of specifying and sharing color maps. The color map facilities of *SunView* are used to control color maps for `cgpixwindd` or `gp1pixwindd` view surfaces (see the *SunView Programmer's Guide*). The user may specify a color map size of 0, in which case a color map of length 2 will be used. Otherwise, *vwsurf.cmapsize* should be a power of 2 between 2 and 256. The user may specify a null color map name, in which case *SunCore* will provide a unique name. Otherwise, *SunCore* will check *vwsurf.cmapname* against the names of the color maps for all windows currently displayed on the physical device on which the new view surface is to appear. If a matching name is found, that color map will be used (even if its size differs from *vwsurf.cmapsize*) and this map is shared among all windows on the device which reference that name. If the user specified a null name or the specified name does not match any current window's color map name, a new color map is allocated with the given size. The indices for each *cgpixwindd* or *gp1pixwindd* view surface's color map run from 0 to *vwsurf.cmapsize*–1.

Currently, one optional string of initialization data may be passed to `initialize_view_surface()`. If *vwsurf.ptr* is non-NULL, it should

point to an array of character pointers, only the first of which will be used. The pointer should point to a string containing position and size information for a Core Tool which may be started up to provide a window for the new view surface. (If the WINDOW_GFX window is taken over by this new view surface and thus no *View Surface Tool* is started, the string will be ignored.) The string should consist of nine integers, separated by commas:

```
"nl,nt,nw,nh,il,it,iw,ih,I"
```

nl, and nt give the initial position of the top left corner of the *View Surface Tool* in its normal form. nw and nh give the initial width and height. The numbers are given in screen coordinates, where (0, 0) is the upper left corner. il, it, iw, and ih give the same initial information for the iconic form of the tool. I is a boolean flag which should be non-zero if the tool is to be started in its iconic form.

B.5. Input Considerations

SunCore uses window system functions to obtain user input from the keyboard and mouse, no matter what mix of raw device view surfaces and window device view surfaces the user has initialized. For purposes of input, a raw device view surface behaves just like a window device view surface; it exists as a window within the window system's data structures, and the user may direct input to the window simply by positioning the mouse over it. The facts that window system input is directed to different windows depending on the location of the mouse and that the mouse position in the window system is reported in the coordinates of the window underlying the mouse have implications for the *SunCore* input functions.

For *SunCore* programs which are invoked from a window within the Suntools window environment, whenever the KEYBOARD device is initialized, `await_keyboard()` will return characters typed when the mouse is located over any initialized view surface (belonging to a single user process) or over the tty subwindow from which the program was started. For programs run from outside a window environment, await_keyboard will return all characters typed on the keyboard, provided the KEYBOARD device is initialized.

The ACM Core specification defines input and output to be completely orthogonal functions. Thus, it is possible to initialize a locator device and read from it without ever initializing a view surface. *SunCore* uses the mouse as the LOCATOR, STROKE, PICK, VALUATOR, and BUTTON devices. The only way *SunCore* can obtain mouse position and button click information to emulate these logical devices is to take input from a window. *SunCore* will return valid data in response to input requests for the LOCATOR, STROKE, PICK, and VALUATOR devices only when the user has associated these devices with an initialized view surface via the `set_echo_surface()` function. Because all *SunCore* view surfaces are instantiations of generic view surface types, there is no default echo surface for any input device. The `set_echo_surface()` function will accept a NULL pointer as its *surface_name* argument to allow the programmer to end the association of an input device with a view surface. Any input device may be echoed on any view surface independently of any other input device.

The input functions `await_any_button_get_locator_2()`, `await_stroke_2()`, `await_pick()`, and

`await_any_button_get_valuator()` will only use mouse input which the user directs to the window which is the echo surface for the indicated LOCATOR, STROKE, PICK, or VALUATOR device. This includes both position and button click input, so that the functions which are terminated by button clicks will terminate only when a button click occurs within the proper window (or a timeout occurs). Which buttons are listened to is still controlled by individually initializing or terminating each BUTTON device.

The user may also use `set_echo_surface()` to choose from which window button clicks should be reported for a BUTTON device when the `await_button()` function is called; alternatively, if the echo surface for a BUTTON device is NULL, `await_button()` will check for button clicks from any view surface associated with a LOCATOR, STROKE, PICK, or VALUATOR device.

Note that the resolution obtained from a LOCATOR, STROKE, PICK, or VALUATOR device is limited by the width and/or height of its echo surface window, since mouse position information is provided by window system input functions in terms of window coordinates.

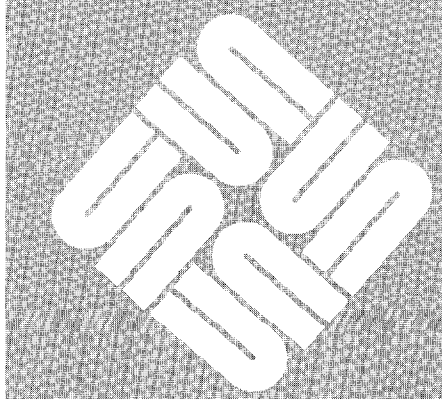
B.6. Notes on Window Device View Surfaces

Graphics primitives drawn on a view surface as part of a temporary segment normally remain visible on the view surface until a new-frame action occurs. For view surfaces which are windows within the Suntools window environment, several user actions can cause the view surface to be redrawn. Such actions include stretching the enclosing tool, exposing a previously obscured portion of the tool, and changing from the iconic form of the tool to the normal form. When the view surface is redrawn in this manner, all output primitives which previously appeared as part of temporary segments will disappear.

When a *SunCore* program is run from a `shelltool(1)`, WINDOW_GFX is normally set to be the tool's tty subwindow. If this window is taken over and blanketed to serve as a view surface, output directed to the tty subwindow (for example, stdout and stderr, including *SunCore* error messages) will not be visible because the blanket window obscures the tty subwindow. When the program terminates or the view surface is terminated, any portion of this output which has not scrolled out of the subwindow will be visible. The fact that the tty subwindow is obscured also means that there is no way to type characters to that window, so that stdin will never see any input. However, if the KEYBOARD device is initialized, special characters, such as interrupt and suspend, typed to the blanket window will be recognized and will have their normal effect on the user process.

C

Alphabetical SunCore C Function Reference



C

Alphabetical SunCore C Function Reference

This appendix contains an alphabetical list of *SunCore* functions and their arguments definitions. *SunCore* programs written in C must contain the statement:

```
#include <usercore.h>
```

at the start of each *SunCore* source file.

C.1. Alphabetical List of C Functions

The list on the following pages is a complete alphabetical list of the functions in *SunCore*.

```
allocate_raster(rptr)
struct {
    int width, height, depth;
    short *bits;
} *rptr;

await_any_button(tim, butnum)
int tim;
int *butnum;

await_any_button_get_locator_2(tim, locnum, butnum, x, y)
int tim, locnum, *butnum;
float *x, *y;

await_any_button_get_valuator(tim, valnum, butnum, val)
int tim, valnum, *butnum;
float *val;

await_keyboard(tim, keynum, string, length)
int tim, keynum;
char *string;
int *length;

await_pick(tim, picknum, segnam, pickid)
int tim;
int picknum, *segnam, *pickid;

await_stroke_2(tim, strokenum, arrsize, xarray, yarray, numxy)
int tim, strokenum, arrsize, *numxy;
float xarray[], yarray[];
```

```
begin_batch_of_updates()

close_retained_segment()

close_temporary_segment()

create_retained_segment(segname)
int segname;

create_temporary_segment()

define_color_indices(surf, i1, i2, red, grn, blu)
struct vwsurf *surf;
int i1, i2;
float *red, *grn, *blu;

delete_all_retained_segments()

delete_retained_segment(segname)
int segname;

deselect_view_surface(surfname)
struct vwsurf *surfname;

end_batch_of_updates()

file_to_raster(rasfid, raster, map)
int rasfid;
struct {
    int width, height, depth;
    short *bits;
} *raster;
struct {
    int type, nbytes;
    char *data;
} *map;

free_raster(rptr)
struct {
    int width, height, depth;
    short *bits;
} *rptr;

get_mouse_state(devclass, devnum, x, y, buttons)
int devclass, devnum;
float *x, *y;
int *buttons;

get_raster(surfname, xmin, xmax, ymin, ymax, xd, yd, raster)
struct vwsurf *surfname;
float xmin, ymin, xmax, ymax;int xd, yd;
struct {
    int width, height, depth;
```

```
    short *bits;
} *raster;

get_view_surface(vsptr, argv)
struct vwsurf *vsptr;
char **argv;

initialize_core(outlev, inlev, dim)
int outlev, inlev, dim;

initialize_device(devclass, devnum)
int devclass, devnum;

initialize_view_surface(surfname, type)
struct vwsurf *surfname;
int type;

inquire_charjust(chjust)
int *chjust;

inquire_charpath_2(dx, dy)
float *dx, *dy;

inquire_charpath_3(dx, dy, dz)
float *dx, *dy, *dz;

inquire_charprecision(chqualty)
int *chqualty;

inquire_charsize(chwidth, cheight)
float *chwidth, *cheight;

inquire_charspace(space)
float *space;

inquire_charup_2(dx, dy)
float *dx, *dy;

inquire_charup_3(dx, dy, dz)
float *dx, *dy, *dz;

inquire_color_indices(surf, i1, i2, red, grn, blu)
struct vwsurf *surf;
int i1, i2;
float *red, *grn, *blu;

inquire_current_position_2(x, y)
float *x, *y;

inquire_current_position_3(x, y, z)
float *x, *y, *z;

inquire_detectability(detectability)
```

```
int *detectability;

inquire_echo(devclass, devnum, echotype)
int devclass, devnum, *echotype;

inquire_echo_position(devclass, devnum, x, y)
int devclass, devnum;
float *x, *y;

inquire_echo_surface(devclass, devnum, surfname)
int devclass, devnum;
struct vwsurf *surfname;

inquire_fill_index(color)
int *color;

inquire_font(font)
int *font;

inquire_highlighting(highlighting)
int *highlighting;

inquire_image_transformation_2(sx, sy, a, tx, ty)
float *sx, *sy, *a, *tx, *ty;

inquire_image_transformation_3(sx, sy, sz, ax, ay, az, tx, ty, tz)
float *sx, *sy, *sz, *ax, *ay, *az, *tx, *ty, *tz;

inquire_image_transformation_type(segtype)
int *segtype;

inquire_image_translate_2(tx, ty)
float *tx, *ty;

inquire_image_translate_3(tx, ty, tz)
float *tx, *ty, *tz;

inquire_inverse_composite_matrix(arrayptr)
float *arrayptr;

inquire_keyboard(keynum, bufsize, istr, pos)
int keynum, *bufsize, *pos;
char *istr;

inquire_line_index(color)
int *color;

inquire_linestyle(linestyl)
int *linestyl;

inquire_linewidth(linwidth)
float *linwidth;
```

```
inquire_locator_2(locnum, x, y)
int locnum;
float *x, *y;

inquire_marker_symbol(mark)
int *mark;

inquire_ndc_space_2(width, height)
float *width, *height;

inquire_ndc_space_3(width, height, depth)
float *width, *height, *depth;

inquire_open_retained_segment(segname)
int *segname;

inquire_open_temporary_segment(open)
int *open;

inquire_pen(pen)
int *pen;

inquire_pick_id(pickid)
int *pickid;

inquire_polygon_edge_style(polyedgstyl)
int *polyedgstyl;

inquire_polygon_interior_style(polyintstyl)
int *polyintstyl;

inquire_primitive_attributes(defprim)
struct {
    int lineindx, fillindx, textindx;
    int linestyl, polyintstyl, polyedgstyl;
    float linwidth;
    int pen, font;
    float charwidth, charheight;
    float charupx, charupy, charupz, charupw;
    float charpathx, charpathy, charpathz, charpathw;
    float charspacex, charspacey, charspacez, charspacew;
    int chjust, chqualty;
    int marker, pickid, rasterop;
} *defprim;

inquire_projection(projection_type, dx_proj, dy_proj, dz_proj)
int *projection_type;

inquire_rasterop(rasterop)
int *rasterop;

inquire_retained_segment_names(listcnt, seglist, segcnt)
int seglist[], listcnt, *segcnt;
```

```
inquire_retained_segment_surfaces(segname, arraycnt, surfaray, surfnum)
int segname, arraycnt;
struct vwsurf surfaray[];
int *surfnum;

inquire_segment_detectability(segname, detectbl)
int segname;
int *detectbl;

inquire_segment_highlighting(segname, highlght)
int segname;
int *highlght;

inquire_segment_image_transformation_2(segname, sx, sy, a, tx, ty)
int segname;
float *sx, *sy, *a, *tx, *ty;

inquire_segment_image_transformation_3(segname, sx, sy, sz, rx, ry, rz, tx, ty, tz)
int segname;
float *sx, *sy, *sz, *rx, *ry, *rz, *tx, *ty, *tz;

inquire_segment_image_translate_2(segname, tx, ty)
int segname;
float *tx, *ty;

inquire_segment_image_translate_3(segname, tx, ty, tz)
int segname;
float *tx, *ty, *tz;

inquire_segment_visibility(segname, visbilty)
int segname;
int *visbilty;

inquire_stroke(strokenum, bufsize, dist, time)
int strokenum, *bufsize, *time;
float *dist;

inquire_text_extent_2(s, dx, dy)
char *s;
float *dx, *dy;

inquire_text_extent_3(s, dx, dy, dz)
char *s;
float *dx, *dy, *dz;

inquire_text_index(color)
int *color;

inquire_valuator(valnum, init, low, high)
int valnum;
float *init, *low, *high;

inquire_view_depth(front_distance, back_distance)
```

```
float *front_distance, *back_distance;

inquire_view_plane_distance(view_distance)
float *view_distance;

inquire_view_plane_normal(dx_norm, dy_norm, dz_norm)
float *dx_norm, *dy_norm, *dz_norm;

inquire_view_reference_point(x_ref, y_ref, z_ref)
float *x_ref, *y_ref, *z_ref;

inquire_view_up_2(dx_up, dy_up)
float *dx_up, *dy_up;

inquire_view_up_3(dx_up, dy_up, dz_up)
float *dx_up, *dy_up, *dz_up;

inquire_viewing_control_parameters(windowclip, frontclip, backclip, type)
int *windowclip, *frontclip, *backclip, *type;

inquire_viewing_parameters(viewparm)
struct {
    float vwrefpt[3];
    float vwplnorm[3];
    float viewdis;
    float frontdis;
    float backdis;
    int projtype;
    float projdir[3];
    float window[4];
    float vwupdir[3];
    float viewport[6];
} *viewparm;

inquire_viewport_2(xmin, xmax, ymin, ymax)
float *xmin, *xmax, *ymin, *ymax;

inquire_viewport_3(xmin, xmax, ymin, ymax, zmin, zmax)
float *xmin, *xmax, *ymin, *ymax, *zmin, *zmax;

inquire_visibility(visibility)
int *visibility;

inquire_window(umin, umax, vmin, vmax)
float *umin, *umax, *vmin, *vmax;

inquire_world_coordinate_matrix_2(arr)
float *arr;

inquire_world_coordinate_matrix_3(arrayptr)
float *arrayptr;

line_abs_2(x, y)
```

```
float x, y;

line_abs_3(x, y, z)
float x, y, z;

line_rel_2(dx, dy)
float dx, dy;

line_rel_3(dx, dy, dz)
float dx, dy, dz;

map_ndc_to_world_2(ndcx, ndcy, wldx, wldy)
float ndcx, ndcy, *wldx, *wldy;

map_ndc_to_world_3(ndcx, ndcy, ndcz, wldx, wldy, wldz)
float ndcx, ndcy, ndcz, *wldx, *wldy, *wldz;

map_world_to_ndc_2(wldx, wldy, ndcx, ndcy)
float wldx, wldy, *ndcx, *ndcy;

map_world_to_ndc_3(wldx, wldy, wldz, ndcx, ndcy, ndcz)
float wldx, wldy, wldz, *ndcx, *ndcy, *ndcz;

marker_abs_2(mx, my)
float mx, my;

marker_abs_3(mx, my, mz)
float mx, my, mz;

marker_rel_2(dx, dy)
float dx, dy;

marker_rel_3(dx, dy, dz)
float dx, dy, dz;

move_abs_2(x, y)
float x, y;

move_abs_3(x, y, z)
float x, y, z;

move_rel_2(dx, dy)
float dx, dy;

move_rel_3(dx, dy, dz)
float dx, dy, dz;

new_frame()

polygon_abs_2(xlist, ylist, n)
float *xlist, *ylist;
short n;
```

```
polygon_abs_3(xlist, ylist, zlist, n)
float *xlist, *ylist, *zlist;
int n;

polygon_rel_2(xlist, ylist, n)
float *xlist, *ylist;
short n;

polygon_rel_3(xlist, ylist, zlist, n)
float *xlist, *ylist, *zlist;
int n;

polyline_abs_2(xcoord, ycoord, n)
float xcoord[], ycoord[];
int n;

polyline_abs_3(xcoord, ycoord, zcoord, n)
float xcoord[], ycoord[], zcoord[];
int n;

polyline_rel_2(xcoord, ycoord, n)
float xcoord[], ycoord[];
int n;

polyline_rel_3(xcoord, ycoord, zcoord, n)
float xcoord[], ycoord[], zcoord[];
int n;

polymarker_abs_2(xcoord, ycoord, n)
float xcoord[], ycoord[];
short n;

polymarker_abs_3(xcoord, ycoord, zcoord, n)
float xcoord[], ycoord[], zcoord[];
int n;

polymarker_rel_2(xcoord, ycoord, n)
float xcoord[], ycoord[];
int n;

polymarker_rel_3(xcoord, ycoord, zcoord, n)
float xcoord[], ycoord[], zcoord[];
int n;

print_error(string, error)
char *string;
int error;

put_raster(srast)
struct {
    int width, height, depth;
    short *bits;
} *srast;
```

```
raster_to_file(raster, map, rasfid, n)
struct {
    int width, height, depth;
    short *bits;
} *raster;
struct {
    int type, nbytes;
    char *data;
} *map;
int rasfid, n;

rename_retained_segment(segname, newname)
int segname, newname;

report_most_recent_error(error)
int *error;

restore_segment(segname, filename)
int segname;
char *filename;

save_segment(segnum, filename)
int segnum;
char *filename;

select_view_surface(surfname)
struct vwsurf *surfname;

set_back_plane_clipping(onoff)
int onoff;

set_charjust(chjust)
int chjust;

set_charpath_2(dx, dy)
float dx, dy;

set_charpath_3(dx, dy, dz)
float dx, dy, dz;

set_charprecision(chqualty)
int chqualty;

set_charsize(chwidth, cheight)
float chwidth, cheight;

set_charspace(space)
float space;

set_charup_2(dx, dy)
float dx, dy;

set_charup_3(dx, dy, dz)
```

```
float dx, dy, dz;

set_coordinate_system_type(type)
int type;

set_detectability(detectability)
int detectability;

set_drag(drag)
int drag;

set_echo(devclass, devnum, echotype)
int devclass, devnum, echotype;

set_echo_group(class, devnum, n, echotype)
int class, devnum[], n, echotype;

set_echo_position(devclass, devnum, x, y)
int devclass, devnum;
float x, y;

set_echo_surface(devclass, devnum, surfname)
int devclass, devnum;
struct vwsurf *surfname;

set_fill_index(color)
int color;

set_font(font)
int font;

set_front_plane_clipping(onoff)
int onoff;

set_highlighting(highlighting)
int highlighting;

set_image_transformation_2(sx, sy, a, tx, ty)
float sx, sy, a, tx, ty;

set_image_transformation_3(sx, sy, sz, ax, ay, az, tx, ty, tz)
float sx, sy, sz, ax, ay, az, tx, ty, tz;

set_image_transformation_type(type)
int type;

set_image_translate_2(tx, ty)
float tx, ty;

set_image_translate_3(tx, ty, tz)
float tx, ty, tz;

set_keyboard(keynum, bufsize, istr, pos)
```

```
int keynum, bufsize, pos;
char *istr;

set_light_direction(dx, dy, dz)
float dx, dy, dz;

set_line_index(color)
int color;

set_linestyle(linestyl)
int linestyl;

set_linewidth(linwidth)
float linwidth;

set_locator_2(locnum, x, y)
int locnum;
float x, y;

set_marker_symbol(mark)
int mark;

set_ndc_space_2(width, height)
float width, height;

set_ndc_space_3(width, height, depth)
float width, height, depth;

set_output_clipping(onoff)
int onoff;

set_pen(pen)
int pen;

set_pick_id(pickid)
int pickid;

set_polygon_edge_style(polyedgstyl)
int polyedgstyl;

set_polygon_interior_style(polyintstyl)
int polyintstyl;

set_primitive_attributes(defprim)
struct {
    int lineindx, fillindx, textindx;
    int linestyl, polyintstyl, polyedgstyl;
    float linwidth;
    int pen, font;
    float charwidth, charheight;
    float charupx, charupy, charupz, charupw;
    float charpathx, charpathy, charpathz, charpathw;
    float charspacex, charspacey, charspacez, charspacew;
```

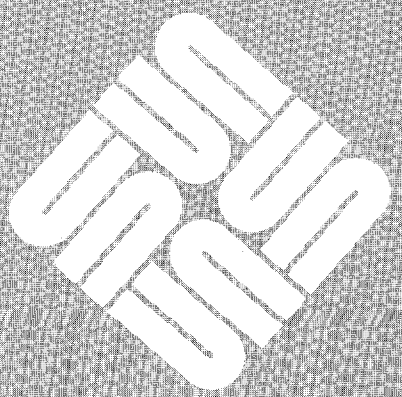
```
    int chjust, chqualty;
    int marker, pickid, rasterop;
} *defprim;

set_projection(projtype, dx, dy, dz)
int projtype;
float dx, dy, dz;

set_rasterop(flag)
int flag;

set_segment_detectability(segname, detectbl)
int segname;
int detectbl;

set_segment_highlighting(segname, highlght)
int segname;
int highlght;

set_segment_image_transformation_2(segname, sx, sy, a, tx, ty)
int segname;
float sx, sy, a, tx, ty;

set_segment_image_translate_2(segname, tx, ty)
int segname;
float tx, ty;

set_segment_image_translate_3(segname, dx, dy, dz)
int segname;
float dx, dy, dz;

set_segment_image_transformation_3(segname, sx, sy, sz, rx, ry, rz, tx, ty, tz)
int segname;
float sx, sy, sz, rx, ry, rz, tx, ty, tz;

set_segment_visibility(segname, visbilty)
int segname;
int visbilty;

set_shading_parameters(amb, dif, spec, flood, bump, hue, style)
float amb, dif, spec, flood, bump;
int hue, style;

set_stroke(strokenum, bufsize, dist, time)
int strokenum, bufsize, time;
float dist;

set_text_index(color)
int color;

set_valuator(valnum, init, low, high)
int valnum;
float init, low, high;
```

```
set_vertex_indices(indxlist, n)
int *indxlist, n;

set_vertex_normals(dxlist, dylist, dzlist, n)
float *dxlist, *dylist, *dzlist;
int n;

set_view_depth(near, far)
float near, far;

set_view_plane_distance(dist)
float dist;

set_view_plane_normal(dx, dy, dz)
float dx, dy, dz;

set_view_reference_point(x, y, z)
float x, y, z;

set_view_up_2(dx, dy)
float dx, dy;

set_view_up_3(dx, dy, dz)
float dx, dy, dz;

set_viewing_parameters(viewparm)
struct {
    float vwrefpt[3];
    float vwplnorm[3];
    float viewdis;
    float frontdis;
    float backdis;
    int projtype;
    float projdir[3];
    float window[4];
    float vwupdir[3];
    float viewport[6];
} *viewparm;

set_viewport_2(xmin, xmax, ymin, ymax)
float xmin, xmax, ymin, ymax;

set_viewport_3(xmin, xmax, ymin, ymax, zmin, zmax)
float xmin, xmax, ymin, ymax, zmin, zmax;

set_visibility(visibility)
int visibility;

set_window(umin, umax, vmin, vmax)
float umin, umax, vmin, vmax;

set_window_clipping(onoff)
int onoff;
```

```
set_world_coordinate_matrix_2(array)
float *array;

set_world_coordinate_matrix_3(array)
float *array;

set_zbuffer_cut(surf, xarr, zarr, n)
struct vwsurf *surf;
float xarr[], zarr[];
int n;

size_raster(surfname, xmin, xmax, ymin, ymax, raster)
struct vwsurf *surfname;
float xmin, ymin, xmax, ymax;
struct {
    int width, height, depth;
    short *bits;
} *raster;

terminate_core()

terminate_device(devclass, devnum)
int devclass, devnum;

terminate_view_surface(surfname)
struct vwsurf *surfname;

text(string)
char *string;
```

D

Using SunCore with Fortran-77 Programs



D

Using SunCore with Fortran-77 Programs

All functions provided in *SunCore* may be called from FORTRAN-77 programs by linking them with the `/usr/lib/libcore77.a` library. This is done by using the *f77* compiler with a command line such as:

```
% f77 -fswitch -o grab grab.f -lcore77 -lcore -lsunwindow -lpixrect -lm
```

where `grab.f` is the FORTRAN source program. The `-fswitch` option will cause the compiler to take advantage of floating point hardware if it is available. Otherwise, the compiler will emulate this floating point support with software. (For more information on floating point options, see Appendix F). Note that `/usr/lib/libcore.a` must be linked with the program (the `-lcore` option), and `/usr/lib/libcore77.a` must come before it (the `-lcore77` option).

Defined constants may be referenced in source programs by including `/usr/include/f77/usercore77.h` In a FORTRAN program, this must be done via a source statement like:

```
include "/usr/include/f77/usercore77.h"
```

This include statement must be in each FORTRAN program unit which uses the defined constants, not just once in each source program file. The default primitive attribute structure PRIMATTS which is provided in <usercore.h> and is described in section 6.1.23 of this manual is not provided in `usercore77.h` because of FORTRAN's restrictions on the ordering of specification statements and data statements.

In the Sun release of FORTRAN-77, names are restricted to sixteen characters in length and may not contain the underline character. For this reason, FORTRAN programs must use abbreviated names to call the corresponding *SunCore* functions. The correspondence between the full *SunCore* names and the FORTRAN names appears later in this appendix. In addition, FORTRAN-77 declarations for all *SunCore* functions appear at the end of this appendix.

D.1. Programming Tips

- The abbreviated names of the *SunCore* functions are less readable than the full length names because the underline character cannot be used in the FORTRAN names. However, since FORTRAN doesn't distinguish between upper-case and lower-case letters in names, upper-case characters can be used to improve readability. There is an example of this later in this appendix.
- Character strings passed from FORTRAN programs to *SunCore* cannot be longer than 256 characters.
- FORTRAN passes all arguments by reference. Although some *SunCore* functions receive arguments by value, the FORTRAN programmer need not worry about this. The interface routines in `/usr/lib/libcore77.a` handle this situation correctly. When in doubt, look at the FORTRAN declarations for *SunCore* subroutines at the end of this appendix.
- *SunCore* uses pointers in some places. For instance, view surface structures contain pointers to device driver functions. Also, the *raster* data type includes a pointer to an array of short's containing the raster data. There are no pointer types in FORTRAN, but there are ways to handle all uses of pointers required to use *SunCore*. For view surface names, the following fragments of C code and FORTRAN code do the same thing:

Table D-1 *Comparison of C and FORTRAN Statements*

C Code	*FORTRAN Code*
`struct vwsurf vsurf = NULL_VWSURF;`	`integer vsurf(VWSURFSIZE)`
`int bw1dd();`	`integer bw1dd` `external bw1dd`
`vsurf.dd = bw1dd;`	`data vsurf /VWSURFSIZE*0/` `vsurf(DDINDEX) = loc(bw1dd)`
`initialize_view_surface(&vsurf, FALSE);`	`call InitializeVwsurf(vsurf, FALSE)`

The constants VWSURFSIZE and DDINDEX are defined in `usercore77.h`. The constant VWSURFNEWFLG is also defined in `usercore77.h`.

bw1dd
The Sun-1 monochrome bitmap display used as a raw device.

bw2dd
The Sun-2 or Sun-3 monochrome bitmap display used as a raw device.

cg1dd
The Sun-1 color graphics display used as a raw device.

cg2dd
The Sun-2 or Sun-3 color graphics display used as a raw device.

cg4dd
The Sun-3/110 color display used as a raw device.

pixwindd
A monochrome (one bit deep) graphics window within the Suntools window environment. This window may appear on either a color or monochrome display.

cgpixwindd
A color graphics window within the Suntools window environment. This window must appear on a color display.

gp1dd
A Sun-2/160 or Sun-3/160 graphics display with a Graphics Processor option.

gp1pixwindd
A color graphics window within the Suntools window environment running on a Sun-2/160 or Sun-3/160 color graphics display with a Graphics Processor option.

Only view surface types *cg1dd, cg2dd, cg4dd, cgpixwindd, gp1dd,* and *gp1pixwindd* support hidden surface removal. In the discussion above, gray scale devices are considered to be color devices.

As shown above, all required pointer manipulation can be done with the FORTRAN `loc` library subroutines, which returns the address of its argument as an integer.

SunCore function arguments which are pointers to structures can be declared as arrays in FORTRAN. For example, the C and FORTRAN declarations of the *SunCore raster* structure are shown below:

C Code	*FORTRAN Code*

`struct {`
`  int width, height, depth;`
`  short *bits;`
`} raster;`
	`integer raster(4)`

Then the following fragments of C and FORTRAN code are equivalent:

C Code	*FORTRAN Code*
`short data[16];`	`integer*2 data(16)`
`raster.width = 16;`	`raster(1) = 16`
`raster.height = 16;`	`raster(2) = 16`
`raster.depth = 1;`	`raster(3) = 1`
`raster.bits = data;`	`raster(4) = loc(data)`

- Some *SunCore* structures contain both and`int`'s `float`'s. For instance, the argument to `inquire_viewing_parameters()` contains both int's and `float`'s. This can be handled in FORTRAN by declaring a REAL array and an INTEGER array which are made to share storage by an EQUIVALENCE

statement. Then following the call to the inquiry function, the REAL components can be accessed by using the REAL array and the INTEGER components accessed via the INTEGER array.

- Since FORTRAN does not distinguish between upper-case and lower-case letters in identifiers, any FORTRAN program unit which includes the `usercore77.h` header file cannot use identifiers with the same spelling as any constant defined in that header file (regardless of case).
- The `filetoraster` and `rastertofile` functions in C take an argument that is a UNIX† file descriptor. The corresponding argument to the FORTRAN functions is a logical unit number (LUN). This unit should be explicitly opened by using the FORTRAN `open` statement. I/O to the opened file should be done *only* via the `filetoraster` and `rastertofile` functions.

D.2. Example Program

This example is the FORTRAN equivalent of the very simple program for drawing a martini glass.

```
include "/usr/include/f77/usercore77.h"

integer vsurf(VWSURFSIZE)
integer pixwindd
external pixwindd
integer InitializeCore, InitializeVwsurf, SelectVwsurf
real glassdx(9), glassdy(9)
data glassdx /-10.0,9.0,0.0,-14.0,30.0,-14.0,0.0,9.0,-10.0/
data glassdy /0.0,1.0,19.0,15.0,0.0,-15.0,-19.0,-1.0, 0.0/
data vsurf /VWSURFSIZE*0/

vsurf(DDINDEX) = loc(pixwindd)
if (InitializeCore(BASIC, NOINPUT, TWOD) .ne. 0) call exit(1)
if (InitializeVwsurf(vsurf, FALSE) .ne. 0) call exit(2)
if (SelectVwsurf(vsurf) .ne. 0) call exit(3)
call SetViewport2(0.125, 0.875, 0.125, 0.75)
call SetWindow(-50.0, 50.0, -10.0, 80.0)
call CreateTempSeg()
call MoveAbs2(0.0, 0.0)
call PolylineRel2(glassdx, glassdy, 9)
call MoveRel2(-12.0, 33.0)
call LineRel2(24.0, 0.0)
call CloseTempSeg()
call sleep(10)
call DeselectVwsurf(vsurf)
call TerminateCore()
end
```

Figure D-1 *FORTRAN Example Program*

† UNIX is a registered trademark of AT&T.

D.3. Correspondence Between C Names and FORTRAN Names

Table D-2 *Correspondence Between C Names and FORTRAN Names*

C Name	*FORTRAN Name*
allocate_raster	allocateraster
await_any_button	awaitanybutton
await_any_button_get_locator_2	awtbuttongetloc2
await_any_button_get_valuator	awtbuttongetval
await_keyboard	awaitkeyboard
await_pick	awaitpick
await_stroke_2	awaitstroke2
begin_batch_of_updates	beginbatchupdate
close_retained_segment	closeretainseg
close_temporary_segment	closetempseg
create_retained_segment	createretainseg
create_temporary_segment	createtempseg
define_color_indices	defcolorindices
delete_all_retained_segments	delallretainsegs
delete_retained_segment	delretainsegment
deselect_view_surface	deselectvwsurf
end_batch_of_updates	endbatchupdate
file_to_raster	filetoraster
free_raster	freeraster
get_mouse_state	getmousestate
get_raster	getraster
initialize_core	initializecore
initialize_device	initializedevice
initialize_view_surface	initializevwsurf
inquire_charjust	inqcharjust
inquire_charpath_2	inqcharpath2
inquire_charpath_3	inqcharpath3
inquire_charprecision	inqcharprecision
inquire_charsize	inqcharsize
inquire_charspace	inqcharspace
inquire_charup_2	inqcharup2
inquire_charup_3	inqcharup3
inquire_color_indices	inqcolorindices
inquire_current_position_2	inqcurrpos2
inquire_current_position_3	inqcurrpos3
inquire_detectability	inqdetectability
inquire_echo	inqecho
inquire_echo_position	inqechoposition
inquire_echo_surface	inqechosurface
inquire_fill_index	inqfillindex
inquire_font	inqfont
inquire_highlighting	inqhighlighting

Table D-2 *Correspondence Between C Names and FORTRAN Names— Continued*

C Name	*FORTRAN Name*
inquire_image_transformation_2	inqimgtransform2
inquire_image_transformation_3	inqimgtransform3
inquire_image_transformation_type	inqimgxformtype
inquire_image_translate_2	inqimgtranslate2
inquire_image_translate_3	inqimgtranslate3
inquire_inverse_composite_matrix	inqinvcompmatrix
inquire_keyboard	inqkeyboard
inquire_line_index	inqlineindex
inquire_linestyle	inqlinestyle
inquire_linewidth	inqlinewidth
inquire_locator_2	inqlocator2
inquire_marker_symbol	inqmarkersymbol
inquire_ndc_space_2	inqndcspace2
inquire_ndc_space_3	inqndcspace3
inquire_open_retained_segment	inqopenretainseg
inquire_open_temporary_segment	inqopentempseg
inquire_pen	inqpen
inquire_pick_id	inqpickid
inquire_polygon_edge_style	inqpolyedgestyle
inquire_polygon_interior_style	inqpolyintrstyle
inquire_primitive_attributes	inqprimattribs
inquire_projection	inqprojection
inquire_rasterop	inqrasterop
inquire_retained_segment_names	inqretainsegname
inquire_retained_segment_surfaces	inqretainsegsurf
inquire_segment_detectability	inqsegdetectable
inquire_segment_highlighting	inqseghighlight
inquire_segment_image_transformation_2	inqsegimgxform2
inquire_segment_image_transformation_3	inqsegimgxform3
inquire_segment_image_transformation_type	inqsegimgxfrmtyp
inquire_segment_image_translate_2	inqsegimgxlate2
inquire_segment_image_translate_3	inqsegimgxlate3
inquire_segment_visibility	inqsegvisibility
inquire_stroke	inqstroke
inquire_text_extent_2	inqtextextent2
inquire_text_extent_3	inqtextextent3
inquire_text_index	inqtextindex
inquire_valuator	inqvaluator
inquire_view_depth	inqviewdepth
inquire_view_plane_distance	inqviewplanedist
inquire_view_plane_normal	inqviewplanenorm
inquire_view_reference_point	inqviewrefpoint
inquire_view_up_2	inqviewup2
inquire_view_up_3	inqviewup3
inquire_viewing_control_parameters	inqvwgcntrlparms
inquire_viewing_parameters	inqviewingparams

Table D-2 *Correspondence Between C Names and FORTRAN Names— Continued*

C Name	*FORTRAN Name*
inquire_viewport_2	inqviewport2
inquire_viewport_3	inqviewport3
inquire_visibility	inqvisibility
inquire_window	inqwindow
inquire_world_coordinate_matrix_2	inqworldmatrix2
inquire_world_coordinate_matrix_3	inqworldmatrix3
line_abs_2	lineabs2
line_abs_3	lineabs3
line_rel_2	linerel2
line_rel_3	linerel3
map_ndc_to_world_2	mapndctoworld2
map_ndc_to_world_3	mapndctoworld3
map_world_to_ndc_2	mapworldtondc2
map_world_to_ndc_3	mapworldtondc3
marker_abs_2	markerabs2
marker_abs_3	markerabs3
marker_rel_2	markerrel2
marker_rel_3	markerrel3
move_abs_2	moveabs2
move_abs_3	moveabs3
move_rel_2	moverel2
move_rel_3	moverel3
new_frame	newframe
polygon_abs_2	polygonabs2
polygon_abs_3	polygonabs3
polygon_rel_2	polygonrel2
polygon_rel_3	polygonrel3
polyline_abs_2	polylineabs2
polyline_abs_3	polylineabs3
polyline_rel_2	polylinerel2
polyline_rel_3	polylinerel3
polymarker_abs_2	polymarkerabs2
polymarker_abs_3	polymarkerabs3
polymarker_rel_2	polymarkerrel2
polymarker_rel_3	polymarkerrel3
print_error	printerror
put_raster	putraster
raster_to_file	rastertofile
rename_retained_segment	renameretainseg
report_most_recent_error	reportrecenterr
restore_segment	restoresegment
save_segment	savesegment
select_view_surface	selectvwsurf
set_back_plane_clipping	setbackclip
set_charjust	setcharjust
set_charpath_2	setcharpath2

Table D-2 *Correspondence Between C Names and FORTRAN Names— Continued*

C Name	*FORTRAN Name*
set_charpath_3	setcharpath3
set_charprecision	setcharprecision
set_charsize	setcharsize
set_charspace	setcharspace
set_charup_2	setcharup2
set_charup_3	setcharup3
set_coordinate_system_type	setcoordsystype
set_detectability	setdetectability
set_drag	setdrag
set_echo	setecho
set_echo_group	setechogroup
set_echo_position	setechoposition
set_echo_surface	setechosurface
set_fill_index	setfillindex
set_font	setfont
set_front_plane_clipping	setfrontclip
set_highlighting	sethighlighting
set_image_transformation_2	setimgtransform2
set_image_transformation_3	setimgtransform3
set_image_transformation_type	setimgxformtype
set_image_translate_2	setimgtranslate2
set_image_translate_3	setimgtranslate3
set_keyboard	setkeyboard
set_light_direction	setlightdirect
set_line_index	setlineindex
set_linestyle	setlinestyle
set_linewidth	setlinewidth
set_locator_2	setlocator2
set_marker_symbol	setmarkersymbol
set_ndc_space_2	setndcspace2
set_ndc_space_3	setndcspace3
set_output_clipping	setoutputclip
set_pen	setpen
set_pick	setpick
set_pick_id	setpickid
set_polygon_edge_style	setpolyedgestyle
set_polygon_interior_style	setpolyintrstyle
set_primitive_attributes	setprimattribs
set_projection	setprojection
set_rasterop	setrasterop
set_segment_detectability	setsegdetectable
set_segment_highlighting	setseghighlight
set_segment_image_transformation_2	setsegimgxform2
set_segment_image_transformation_3	setsegimgxform3
set_segment_image_translate_2	setsegimgxlate2
set_segment_image_translate_3	setsegimgxlate3

Table D-2 *Correspondence Between C Names and FORTRAN Names— Continued*

C Name	*FORTRAN Name*
set_segment_visibility	setsegvisibility
set_shading_parameters	setshadingparams
set_stroke	setstroke
set_text_index	settextindex
set_valuator	setvaluator
set_vertex_indices	setvertexindices
set_vertex_normals	setvertexnormals
set_view_depth	setviewdepth
set_view_plane_distance	setviewplanedist
set_view_plane_normal	setviewplanenorm
set_view_reference_point	setviewrefpoint
set_viewport_2	setviewport2
set_viewport_3	setviewport3
set_view_up_2	setviewup2
set_view_up_3	setviewup3
set_viewing_parameters	setviewingparams
set_visibility	setvisibility
set_window	setwindow
set_window_clipping	setwindowclip
set_world_coordinate_matrix_2	setworldmatrix2
set_world_coordinate_matrix_3	setworldmatrix3
set_zbuffer_cut	setzbuffercut
size_raster	sizeraster
terminate_core	terminatecore
terminate_device	terminatedevice
terminate_view_surface	terminatevwsurf
text	text

D.4. FORTRAN Interfaces to SunCore

Note: Although all *SunCore* procedures are declared here as functions, each may also be called as a subroutine if the user does not want to check the returned value.

```
integer function allocateraster(raster)
integer raster(4)

integer function awaitanybutton(time, buttonnum)
integer time, buttonnum

integer function awtbuttongetloc2(time, locatornum, buttonnum, x, y)
integer time, locatornum, buttonnum
real x, y

integer function awtbuttongetval(time,valuatornum,buttonnum,value)
integer time, valuatornum, buttonnum
real value

integer function awaitkeyboard(time, keyboardnum, inputstring, length)
```

```
integer time, keyboardnum
character*(*) inputstring
integer length

integer function awaitpick(time, picknum, segname, pickid)
integer time, picknum, segname, pickid

integer function awaitstroke2(time, strokenum, arraysize, xarray, yarray, n)
integer time, strokenum, arraysize
real xarray, yarray
integer n

integer function beginbatchupdate()

integer function closeretainseg()

integer function closetempseg()

integer function createretainseg(segname)
integer segname

integer function createtempseg()

integer function defcolorindices(surfacename, i1, i2, red, green, blue)
integer surfacename(*)
integer i1, i2
real red(*), green(*), blue(*)

integer function delallretainsegs()

integer function delretainsegment(segname)
integer segname

integer function deselectvwsurf(surfacename)
integer surfacename(*)

integer function endbatchupdate()

integer function filetoraster(rasfid, raster, map)
integer rasfid
integer raster(4)
integer map(3)

integer function freeraster(raster)
integer raster(4)

integer function getmousestate(devclass, devnum, x, y, buttons)
integer devclass, devnum
real x, y
integer buttons

integer function getraster(surfacename, xmin, xmax, ymin, ymax, xd, yd, raster)
integer surfacename(*)
```

```
real xmin, xmax, ymin, ymax
integer xd, yd
integer raster(4)

integer function initializecore(outputlevel, inputlevel, dimension)
integer outputlevel, inputlevel, dimension

integer function initializedevice(deviceclass, devicenum)
integer deviceclass, devicenum

integer function initializevwsurf(surfacename, type)
integer surfacename(*)
integer type

integer function inqcharjust(just)
integer just

integer function inqcharpath2(dx, dy)
real dx, dy

integer function inqcharpath3(dx, dy, dz)
real dx, dy, dz

integer function inqcharprecision(charprecision)
integer charprecision

integer function inqcharsize(charwidth, charheight)
real charwidth, charheight

integer function inqcharspace(charspace)
real charspace

integer function inqcharup2(dx, dy)
real dx, dy

integer function inqcharup3(dx, dy, dz)
real dx, dy, dz

integer function inqcolorindices(surfacename, i1, i2, red, green, blue)
integer surfacename(*)
integer i1, i2
real red(*), green(*), blue(*)

integer function inqcurrpos2(x, y)
real x, y

integer function inqcurrpos3(x, y, z)
real x, y, z

integer function inqdetectability(detectability)
integer detectability

integer function inqecho(deviceclass, devicenum, echotype)
```

```
integer deviceclass, devicenum, echotype

integer function inqechoposition(deviceclass, devicenum, echox, echoy)
integer deviceclass, devicenum
real echox, echoy

integer function inqechosurface(deviceclass, devicenum, surfacename)
integer deviceclass, devicenum
integer surfacename(*)

integer function inqfillindex(index)
integer index

integer function inqfont(font)
integer font

integer function inqhighlighting(highlighting)
integer highlighting

integer function inqimgtransform2(sx, sy, a, tx, ty)
real sx, sy, a, tx, ty

integer function inqimgtransform3(sx, sy, sz, ax, ay, az, tx, ty, tz)
real sx, sy, sz, ax, ay, az, tx, ty, tz

integer function inqimgxformtype(type)
integer type

integer function inqimgtranslate2(tx, ty)
real tx, ty

integer function inqimgtranslate3(tx, ty, tz)
real tx, ty, tz

integer function inqinvcompmatrix(array)
real array(4,4)

integer function inqkeyboard(keyboardnum, buffersize, initstring, initcursor)
integer keyboardnum, buffersize
character*(*) initstring
integer initcursor

integer function inqlineindex(index)
integer index

integer function inqlinestyle(linestyle)
integer linestyle

integer function inqlinewidth(linewidth)
real linewidth

integer function inqlocator2(locatornum, x, y)
integer locatornum
```

```
real x, y

integer function inqmarkersymbol(symbol)
integer symbol

integer function inqndcspace2(width, height)
real width, height

integer function inqndcspace3(width, height, depth)
real width, height, depth

integer function inqopenretainseg(segname)
integer segname

integer function inqopentempseg(open)
integer open

integer function inqpen(pen)
integer pen

integer function inqpickid(pickid)
integer pickid

integer function inqpolyedgestyle(style)
integer style

integer function inqpolyintrstyle(style)
integer style

integer function inqprimattribs(primattr)
integer primattr(28)
```

Note: The actual argument in the calling program corresponding to primattr should be an array which can be referenced both as a real array and as an integer array in order to access both integer valued and real valued primitive attributes. This can be done using the `equivalence` statement.

```
integer function inqprojection(projection, dxproj, dyproj, dzproj)
integer projection real dxproj, dyproj, dzproj

integer function inqrasterop(rop)
integer rop

integer function inqretainsegname(arraysize, namearray, numberofsegments)
integer arraysize, namearray(*), numberofsegments

integer function inqretainsegsurf(segname, arraysize, vwsurfarray, numsurf)
integer segname, arraysize
integer vwsurfarray(*)
integer numsurf
```

Note: *arraysize* should give the number of view surface structures which can be held in *vwsurfarray*. Each structure requires VWSURFSIZE elements of *vwsurfarray*.

```
integer function inqsegdetectable(segname, detectability)
integer segname, detectability
```

```
integer function inqseghighlight(segname, highlighting)
integer segname, highlighting

integer function inqsegimgxform2(segname, sx, sy, a, tx, ty)
integer segname
real sx, sy, a, tx, ty

integer function inqsegimgxform3(segname, sx, sy, sz, ax, ay, az, tx, ty, tz)
integer segname
real sx, sy, sz, ax, ay, az, tx, ty, tz

integer function inqsegimgxfrmtyp(segname, type)
integer segname, type

integer function inqsegimgxlate2(segname, tx, ty)
integer segname
real tx, ty

integer function inqsegimgxlate3(segname, tx, ty, tz)
integer segname
real tx, ty, tz

integer function inqsegvisibility(segname, visibility)
integer segname, visibility

integer function inqstroke(strokenum, bufsize, dist, time)
integer strokenum, bufsize
real dist
integer time

integer function inqtextextent2(string, dx, dy)
character*(*) string
real dx, dy

integer function inqtextextent3(string, dx, dy, dz)
character*(*) string
real dx, dy, dz

integer function inqtextindex(index)
integer index

integer function inqvaluator(valuatornum, initialvalue, low, high)
integer valuatornum
real initialvalue, low, high

integer function inqviewdepth(frontdistance, backdistance)
real frontdistance, backdistance

integer function inqviewplanedist(viewdistance)
real viewdistance

integer function inqviewplanenorm(dxnorm, dynorm, dznorm)
real dxnorm, dynorm, dznorm
```

```
integer function inqviewrefpoint(x, y, z)
real x, y, z

integer function inqviewup2(dxup, dyup)
real dxup, dyup

integer function inqviewup3(dxup, dyup, dzup)
real dxup, dyup, dzup

integer function inqvwgcntrlparms(windowclip, frontclip, backclip, type)
integer windowclip, frontclip, backclip, type

integer function inqviewingparams(viewparams)
real viewparams(26)
```

Note: The actual argument in the calling program corresponding to *viewparams* should be an array which can be referenced both as a real array and as an integer array in order to access both integer valued and real valued viewing parameters. This can be done using the `equivalence` statement.

```
integer function inqviewport2(xmin, xmax, ymin, ymax)
real xmin, xmax, ymin, ymax

integer function inqviewport3(xmin, xmax, ymin, ymax, zmin, zmax)
real xmin, xmax, ymin, ymax, zmin, zmax

integer function inqvisibility(visibility)
integer visibility

integer function inqwindow(umin, umax, vmin, vmax)
real umin, umax, vmin, vmax

integer function inqworldmatrix2(array)
real array(3,3)

integer function inqworldmatrix3(array)
real array(4,4)

integer function lineabs2(x, y)
real x, y

integer function lineabs3(x, y, z)
real x, y, z

integer function linerel2(dx, dy)
real dx, dy

integer function linerel3(dx, dy, dz)
real dx, dy, dz

integer function mapndctoworld2(ndcx, ndcy, wldx, wldy)
real ndcx, ndcy, wldx, wldy

integer function mapndctoworld3(ndcx, ndcy, ndcz, wldx, wldy, wldz)
real ndcx, ndcy, ndcz, wldx, wldy, wldz
```

```
integer function mapworldtondc2(wldx, wldy, ndcx, ndcy)
real wldx, wldy, ndcx, ndcy

integer function mapworldtondc3(wldx, wldy, wldz, ndcx, ndcy, ndcz)
real wldx, wldy, wldz, ndcx, ndcy, ndcz

integer function markerabs2(x, y)
real x, y

integer function markerabs3(x, y, z)
real x, y, z

integer function markerrel2(dx, dy)
real dx, dy

integer function markerrel3(dx, dy, dz)
real dx, dy, dz

integer function moveabs2(x, y)
real x, y

integer function moveabs3(x, y, z)
real x, y, z

integer function moverel2(dx, dy)
real dx, dy

integer function moverel3(dx, dy, dz)
real dx, dy, dz

integer function newframe()

integer function polygonabs2(xarray, yarray, n)
real xarray(*), yarray(*)
integer n

integer function polygonabs3(xarray, yarray, zarray, n)
real xarray(*), yarray(*), zarray(*)
integer n

integer function polygonrel2(dxarray, dyarray, n)
real dxarray(*), dyarray(*)
integer n

integer function polygonrel3(dxarray, dyarray, dzarray, n)
real dxarray(*), dyarray(*), dzarray(*)
integer n

integer function polylineabs2(xarray, yarray, n)
real xarray(*), yarray(*)
integer n

integer function polylineabs3(xarray, yarray, zarray, n)
```

```
real xarray(*), yarray(*), zarray(*)
integer n

integer function polylinerel2(dxarray, dyarray, n)
real dxarray(*), dyarray(*)
integer n

integer function polylinerel3(dxarray, dyarray, dzarray, n)
real dxarray(*), dyarray(*), dzarray(*)
integer n

integer function polymarkerabs2(xarray, yarray, n)
real xarray(*), yarray(*)
integer n

integer function polymarkerabs3(xarray, yarray, zarray, n)
real xarray(*), yarray(*), zarray(*)
integer n

integer function polymarkerrel2(dxarray, dyarray, n)
real dxarray(*), dyarray(*)
integer n

integer function polymarkerrel3(dxarray, dyarray, dzarray, n)
real dxarray(*), dyarray(*), dzarray(*)
integer n

integer function printerror(message, errornum)
character*(*) message
integer errornum

integer function putraster(raster)
integer raster(4)

integer function rastertofile(raster, map, rasfid, n)
integer raster(4)
integer map(3)
integer rasfid, n

integer function renameretainseg(segname, newname)
integer segname, newname

integer function reportrecenterr(errornum)
integer errornum

integer function restoresegment(segname, filename)
integer segname
character*(*) filename

integer function savesegment(segname, filename)
integer segname
character*(*) filename
```

```
integer function selectvwsurf(surfacename)
integer surfacename(*)

integer function setbackclip(onoff)
integer onoff

integer function setcharjust(just)
integer just

integer function setcharpath2(dx, dy)
real dx, dy

integer function setcharpath3(dx, dy, dz)
real dx, dy, dz

integer function setcharprecision(charprecision)
integer charprecision

integer function setcharsize(charwidth, charheight)
real charwidth, charheight

integer function setcharspace(charspace)
real charspace

integer function setcharup2(dx, dy)
real dx, dy

integer function setcharup3(dx, dy, dz)
real dx, dy, dz

integer function setcoordsystype(type)
integer type

integer function setdetectability(detectability)
integer detectability

integer function setdrag(mode)
integer mode

integer function setecho(deviceclass, devicenum, echotype)
integer deviceclass, devicenum, echotype

integer function setechogroup(deviceclass, devicenumarray, n, echotype)
integer deviceclass, devicenumarray(*), n, echotype

integer function setechoposition(deviceclass, devicenum, echox, echoy)
integer deviceclass, devicenum
real echox, echoy

integer function setechosurface(deviceclass, devicenum, surfacename)
integer deviceclass, devicenum
integer surfacename(*)
```

```
integer function setfillindex(index)
integer index

integer function setfont(font)
integer font

integer function setfrontclip(onoff)
integer onoff

integer function sethighlighting(highlighting)
integer highlighting

integer function setimgtransform2(sx, sy, a, tx, ty)
real sx, sy, a, tx, ty

integer function setimgtransform3(sx, sy, sz, ax, ay, az, tx, ty, tz)
real sx, sy, sz, ax, ay, az, tx, ty, tz

integer function setimgxformtype(type)
integer type

integer function setimgtranslate2(tx, ty)
real tx, ty

integer function setimgtranslate3(tx, ty, tz)
real tx, ty, tz

integer function setkeyboard(keyboardnum, buffersize, initstring, initcursor)
integer keyboardnum, buffersize
character*(*) initstring
integer initcursor

integer function setlightdirect(dx, dy, dz)
real dx, dy, dz

integer function setlineindex(index)
integer index

integer function setlinestyle(linestyle)
integer linestyle

integer function setlinewidth(linewidth)
real linewidth

integer function setlocator2(locatornum, x, y)
integer locatornum
real x, y

integer function setmarkersymbol(symbol)
integer symbol

integer function setndcspace2(width, height)
real width, height
```

```
integer function setndcspace3(width, height, depth)
real width, height, depth

integer function setoutputclip(onoff)
integer onoff

integer function setpen(pen)
integer pen

integer function setpick(picknum, aperture)
integer picknum
real aperture

integer function setpickid(pickid)
integer pickid

integer function setpolyedgestyle(style)
integer style

integer function setpolyintrstyle(style)
integer style

integer function setprimattribs(primattr)
integer primattr(28)
```

Note: The actual argument in the calling program corresponding to *primattr* should be an array which can be referenced both as a real array and as an integer array in order to access both integer valued and real valued primitive attributes. This can be done using the `equivalence` statement.

```
integer function setprojection(projection, dxproj, dyproj, dzproj)
integer projection
real dxproj, dyproj, dzproj

integer function setrasterop(rop)
integer rop

integer function setsegdetectable(segname, detectability)
integer segname, detectability

integer function setseghighlight(segname, highlighting)
integer segname, highlighting

integer function setsegimgxform2(segname, sx, sy, a, tx, ty)
integer segname
real sx, sy, a, tx, ty

integer function setsegimgxform3(segname, sx, sy, sz, ax, ay, az, tx, ty, tz)
integer segname
real sx, sy, sz, ax, ay, az, tx, ty, tz

integer function setsegimgxlate2(segname, tx, ty)
integer segname
real tx, ty
```

```
integer function setsegimgxlate3(segname, tx, ty, tz)
integer segname
real tx, ty, tz

integer function setsegvisibility(segname, visibility)
integer segname, visibility

integer function setshadingparams(ambient, diffuse, specular, flood, bump, hue, style)
real ambient, diffuse, specular, flood, bump
integer hue, style

integer function setstroke(strokenum, buffersize, distance, time)
integer strokenum, buffersize
real distance
integer time

integer function settextindex(index)
integer index

integer function setvaluator(valuatornum, initialvalue, low, high)
integer valuatornum
real initialvalue, low, high

integer function setvertexindices(colorindexlist, n)
integer colorindexlist(*), n

integer function setvertexnormals(xlist, ylist, zlist, n)
real xlist(*), ylist(*), zlist(*)
integer n

integer function setviewdepth(frontdistance, backdistance)
real frontdistance, backdistance

integer function setviewplanedist(distance)
real distance

integer function setviewplanenorm(dxnorm, dynorm, dznorm)
real dxnorm, dynorm, dznorm

integer function setviewport2(xmin, xmax, ymin, ymax)
real xmin, xmax, ymin, ymax

integer function setviewport3(xmin, xmax, ymin, ymax, zmin, zmax)
real xmin, xmax, ymin, ymax, zmin, zmax

integer function setviewrefpoint(x, y, z)
real x, y, z

integer function setviewup2(dx, dy)
real dx, dy

integer function setviewup3(dx, dy, dz)
real dx, dy, dz
```

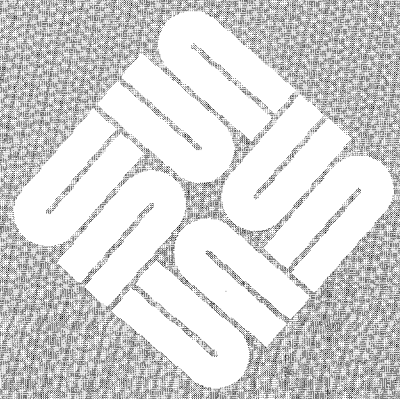
```
integer function setviewingparams(viewparams)
real viewparams(26)
```

Note: The actual argument in the calling program corresponding to *viewparams* should be an array which can be referenced both as a real array and as an integer array in order to access both integer valued and real valued viewing parameters. This can be done using the `equivalence` statement.

```
integer function setvisibility(visibility)
integer visibility

integer function setwindow(umin, umax, vmin, vmax)
real umin, umax, vmin, vmax

integer function setwindowclip(onoff)
integer onoff

integer function setworldmatrix2(array)
real array(3,3)

integer function setworldmatrix3(array)
real array(4,4)

integer function setzbuffercut(surfacename, xlist, zlist, n)
integer surfacename(*)
real xlist(*), zlist(*)
integer n

integer function sizeraster(surfacename, xmin, xmax, ymin, ymax, raster)
integer surfacename(*)
real xmin, xmax, ymin, ymax
integer raster(4)

integer function terminatecore()

integer function terminatedevice(deviceclass, devicenum)
integer deviceclass, devicenum

integer function terminatevwsurf(surfacename)
integer surfacename(*)

integer function text(string)
character*(*) string
```

E

Using SunCore with Pascal Programs



E

Using SunCore with Pascal Programs

All functions provided in *SunCore* may be called from Pascal programs by linking them with the `/usr/lib/libcorepas.a` library by using the Pascal compiler with a command line of the form:

```
% pc -fswitch -o grab grab.p -lcorepas -lcore -lsunwindow -lpixrect -lm
```

where `grab.p` is the Pascal source program. The `-fswitch` option will cause the compiler to take advantage of floating point hardware if it is available. Otherwise, the compiler will emulate this floating point support with software. (For more information on floating point options, see Appendix F). Note that `/usr/lib/libcore.a` must be linked with the program (the `-lcore` option), and `/usr/lib/libcorepas.a` must come before it (the `-lcorepas` option).

E.1. Programming Requirements

The files `typedefspas.h`, `usercorepas.h`, `devincpas.h` and `sunpas.h` from the `/usr/include/pascal` directory must be included in the user's source code to provide the necessary declarations for the Pascal interface to *SunCore*. Pascal programs which call *SunCore* functions must place these include files in the most global declaration section of the program:

```
    program example (input,output)

#include '/usr/include/pascal/usercorepas.h'
#include '/usr/include/pascal/typedefspas.h'

    var
        {user declarations}

#include '/usr/include/pascal/devincpas.h'
#include '/usr/include/pascal/sunpas.h'
```

If the Pascal program is composed of separately compiled files, these include statements must be in each Pascal file which uses *SunCore* functions and the corresponding defined constants. Defined constants for *SunCore* (see section on *Useful Constants* in the introduction to this manual) are set in the file `/usr/include/pascal/usercorepas.h`. The default primitive attribute structure PRIMATTS provided in `usercore.h` and described in the section describing *set_primitive_attributes* is not provided in `usercorepas.h`.

The Sun release of Pascal does not support the passing of variable length arrays as arguments in function or procedure calls. Therefore, fixed length arrays which are compatible with the *SunCore*-Pascal interface are declared as predefined types in the `typedefspas.h` file (see the *Declarations* section of this appendix). The length of these arrays in 256. The length of character strings passed from Pascal programs to *SunCore* must also be 256 characters.

The correspondence between the full *SunCore* names and the Pascal names appears in the Function Declarations section of this appendix. To provide a mechanism for returning the status of calls to *SunCore* routines, all *SunCore* routines must be called as functions from Pascal. Finally, although most *SunCore* functions use floats (32-bit reals), Pascal uses 64-bit reals. However, the Pascal programmer is only required to provide reals. *SunCore* functions which have structures as their arguments have corresponding predefined types in Pascal (see the *Type Declarations* section of this appendix).

Routines Using View Surface Names

View surface names in *SunCore* are structures containing pointers to device driver routines. The device driver names are supplied by the include file `devincpas.h`. The user may then simply use one of the names listed in Table E-1:

Table E-1 *Viewsurface Types*

Symbol	*Description*
`bw1dd`	Sun-1 monochrome display
`bw2dd`	Sun-2 monochrome display
`cg1dd`	Sun-1 color display
`cg2dd`	Sun-2 color display
`cg4dd`	Sun-3/110 color display
`gp1dd`	Graphics Processor
`pixwindd`	windows on the Sun-1 monochrome display
`cgpixwindd`	windows on a color display
`gp1pixwindd`	windows with the Graphics Processor

The `pasloc` function (provided in the *SunCore*-Pascal interface) transforms the function corresponding to the device driver into an integer which can then be inserted in the appropriate place in the device driver structure (see following example).

Table E-2 *Comparison of C and Pascal Statements*

C Code	*Pascal Code*
`struct vwsurf dsurf = NULL_VWSURF;`	`var`
`int bw1dd();`	`dsurf:vwsurf;`
	`tstr:vwsurfst;`
`.`	`.`
`.`	`.`
`.`	`.`
	`tstr := '  ';`
`dsurf.dd = bw1dd;`	`dsurf.dd := pasloc(bw1dd);`
	`dsurf.screenname := tstr;`
	`dsurf.windowname := tstr;`
	`dsurf.windowfd := 0;`
	`dsurf.instance := 0;`
	`dsurf.cmapsize := 0;`
	`dsurf.cmapname := tstr;`
	`dsurf.flags := 0;`
	`dsurf.ptr := 0;`
`initialize_view_surface(&dsurf, FALSE);`	`x := InitializeVwsurf(dsurf, FALSE);`

Assigning a literal string of two spaces (blanks) to the *tstr* variable will initialize the character array to all spaces.

Routines Using Rasters and Colormaps

For uses of *SunCore* functions which have rasters or colormaps as arguments which do not involve arithmetic direct manipulation by the programmer (for example, writing a raster to a file), the following restrictions on the functions do not apply and the programmer is only required to call the function. *SunCore* raster and colormap structures contain pointers to variable length data (that is, dynamic arrays). The *SunCore*-Pascal interface declares these varaibles as integers.

Pascal programmers wishing to alter the contents of the colormap or raster data within a program can write a C function which uses the pointer value returned in Pascal to copy the information into a fixed-length array. Arithmetic operations can then be performed on the data using conventional Pascal statements. The programmer can then write another C function to copy the information back into the array pointed to by the pointer returned by the *SunCore*-Pascal interface. These C functions are not provided because the size of the fixed-length array will vary greatly among different applications. Therefore, the individual Pascal programmer must decide how large an array to declare for each application.

E.2. Example Program

The use of the *SunCore*-Pascal interface is illustrated by showing the text of a program for drawing the martini glass used in previous tutorial examples.

Figure E-1 *Pascal Example Program*

```
    program martiniglass (input,output);

#include '/usr/include/pascal/usercorepas.h';
```

```
#include '/usr/include/pascal/typedefspas.h';

    var
         glassdx, glassdy: parr {type parr is an array of reals of
                    length 256 declared in typedefs.h};
         x:integer;
         dsurf:vwsurf;
         tstr:vsurfst;
         function sleep(x:integer):integer; external;
#include '/usr/include/pascal/sunpas.h';
#include '/usr/include/pascal/devincpas.h';

    procedure loaddata;
        begin
            glassdx[1] := -10.0;  glassdy[1] := 0.0;
            glassdx[2] := 9.0;    glassdy[2] := 1.0;
            glassdx[3] := 0.0;    glassdy[3] := 19.0;
            glassdx[4] := -14.0;  glassdy[4] := 15.0;
            glassdx[5] := 30.0;   glassdy[5] := 0.0;
            glassdx[6] := -14.0;  glassdy[6] := -15.0;
            glassdx[7] := 0.0;    glassdy[7] := -19.0;
            glassdx[8] := 9.0;    glassdy[8] := -1.0;
            glassdx[9] := -10.0;  glassdy[9] := 0.0;
        end;

    begin {main program}
    tstr := '  ';
    dsurf.screenname := tstr;
    dsurf.windowname := tstr;
    dsurf.windowfd := 0;
    dsurf.dd := pasloc(pixwindd);
    dsurf.instance := 0;
    dsurf.cmapsize := 0;
    dsurf.cmapname := tstr;
    dsurf.flags := 0;
        if (initializecore(BASIC, NOINPUT, TWOD) <> 0) then
        writeln (' error 1')
        else
          if (initializevwsurf(dsurf, FALSE) <> 0) then
        writeln (' error 2')
          else
            if (selectvwsurf(dsurf) <> 0) then
          writeln (' error 3')
        else
               x := setviewport2(0.125, 0.875, 0.125, 0.75);
               x := setwindow(-50.0, 50.0, -10.0, 80.0);
               x := createtempseg;
           x := moveabs2(0.0, 0.0);
           loaddata;
               x := polylinerel2(glassdx, glassdy,9);
               x := moverel2(-12.0, 33.0);
               x := linerel2(24.0, 0.0);
               x := closetempseg;
```

```
            x := sleep(10);
            x := deselectvwsurf(dsurf);
            x := terminatecore;
end.
```

E.3. Correspondence Between C Names and Pascal Names

Table E-3 *Correspondence Between C Names and Pascal Names*

C Name	*Pascal Name*
allocate_raster	allocateraster
await_any_button	awaitanybutton
await_any_button_get_locator_2	awtbuttongetloc2
await_any_button_get_valuator	awtbuttongetval
await_keyboard	awaitkeyboard
await_pick	awaitpick
await_stroke_2	awaitstroke2
begin_batch_of_updates	beginbatchupdate
close_retained_segment	closeretainseg
close_temporary_segment	closetempseg
create_retained_segment	createretainseg
create_temporary_segment	createtempseg
define_color_indices	defcolorindices
delete_all_retained_segments	delallretainsegs
delete_retained_segment	delretainsegment
deselect_view_surface	deselectvwsurf
end_batch_of_updates	endbatchupdate
file_to_raster	filetoraster
free_raster	freeraster
get_mouse_state	getmousestate
get_raster	getraster
initialize_core	initializecore
initialize_device	initializedevice
initialize_view_surface	initializevwsurf
inquire_charjust	inqcharjust
inquire_charpath_2	inqcharpath2
inquire_charpath_3	inqcharpath3
inquire_charprecision	inqcharprecision
inquire_charsize	inqcharsize
inquire_charspace	inqcharspace
inquire_charup_2	inqcharup2
inquire_charup_3	inqcharup3
inquire_color_indices	inqcolorindices
inquire_current_position_2	inqcurrpos2
inquire_current_position_3	inqcurrpos3
inquire_detectability	inqdetectability

Table E-3 *Correspondence Between C Names and Pascal Names— Continued*

C Name	*Pascal Name*
inquire_echo	inqecho
inquire_echo_position	inqechoposition
inquire_echo_surface	inqechosurface
inquire_fill_index	inqfillindex
inquire_font	inqfont
inquire_highlighting	inqhighlighting
inquire_image_transformation_2	inqimgtransform2
inquire_image_transformation_3	inqimgtransform3
inquire_image_transformation_type	inqimgxformtype
inquire_image_translate_2	inqimgtranslate2
inquire_image_translate_3	inqimgtranslate3
inquire_inverse_composite_matrix	inqinvcompmatrix
inquire_keyboard	inqkeyboard
inquire_line_index	inqlineindex
inquire_linestyle	inqlinestyle
inquire_linewidth	inqlinewidth
inquire_locator_2	inqlocator2
inquire_marker_symbol	inqmarkersymbol
inquire_ndc_space_2	inqndcspace2
inquire_ndc_space_3	inqndcspace3
inquire_open_retained_segment	inqopenretainseg
inquire_open_temporary_segment	inqopentempseg
inquire_pen	inqpen
inquire_pick_id	inqpickid
inquire_polygon_edge_style	inqpolyedgestyle
inquire_polygon_interior_style	inqpolyintrstyle
inquire_primitive_attributes	inqprimattribs
inquire_projection	inqprojection
inquire_rasterop	inqrasterop
inquire_retained_segment_names	inqretainsegname
inquire_retained_segment_surfaces	inqretainsegsurf
inquire_segment_detectability	inqsegdetectable
inquire_segment_highlighting	inqseghighlight
inquire_segment_image_transformation_2	inqsegimgxform2
inquire_segment_image_transformation_3	inqsegimgxform3
inquire_segment_image_transformation_type	inqsegimgxfrmtyp
inquire_segment_image_translate_2	inqsegimgxlate2
inquire_segment_image_translate_3	inqsegimgxlate3
inquire_segment_visibility	inqsegvisibility
inquire_stroke	inqstroke
inquire_text_extent_2	inqtextextent2
inquire_text_extent_3	inqtextextent3
inquire_text_index	inqtextindex
inquire_valuator	inqvaluator
inquire_view_depth	inqviewdepth
inquire_view_plane_distance	inqviewplanedist

Table E-3 *Correspondence Between C Names and Pascal Names— Continued*

C Name	*Pascal Name*
inquire_view_plane_normal	inqviewplanenorm
inquire_view_reference_point	inqviewrefpoint
inquire_view_up_2	inqviewup2
inquire_view_up_3	inqviewup3
inquire_viewing_control_parameters	inqvwgcntrlparms
inquire_viewing_parameters	inqviewingparams
inquire_viewport_2	inqviewport2
inquire_viewport_3	inqviewport3
inquire_visibility	inqvisibility
inquire_window	inqwindow
inquire_world_coordinate_matrix_2	inqworldmatrix2
inquire_world_coordinate_matrix_3	inqworldmatrix3
line_abs_2	lineabs2
line_abs_3	lineabs3
line_rel_2	linerel2
line_rel_3	linerel3
map_ndc_to_world_2	mapndctoworld2
map_ndc_to_world_3	mapndctoworld3
map_world_to_ndc_2	mapworldtondc2
map_world_to_ndc_3	mapworldtondc3
marker_abs_2	markerabs2
marker_abs_3	markerabs3
marker_rel_2	markerrel2
marker_rel_3	markerrel3
move_abs_2	moveabs2
move_abs_3	moveabs3
move_rel_2	moverel2
move_rel_3	moverel3
new_frame	newframe
polygon_abs_2	polygonabs2
polygon_abs_3	polygonabs3
polygon_rel_2	polygonrel2
polygon_rel_3	polygonrel3
polyline_abs_2	polylineabs2
polyline_abs_3	polylineabs3
polyline_rel_2	polylinerel2
polyline_rel_3	polylinerel3
polymarker_abs_2	polymarkerabs2
polymarker_abs_3	polymarkerabs3
polymarker_rel_2	polymarkerrel2
polymarker_rel_3	polymarkerrel3
print_error	printerror
put_raster	putraster
raster_to_file	rastertofile
rename_retained_segment	renameretainseg
report_most_recent_error	reportrecenterr

Table E-3 *Correspondence Between C Names and Pascal Names— Continued*

C Name	*Pascal Name*
restore_segment	restoresegment
save_segment	savesegment
select_view_surface	selectvwsurf
set_back_plane_clipping	setbackclip
set_charjust	setcharjust
set_charpath_2	setcharpath2
set_charpath_3	setcharpath3
set_charprecision	setcharprecision
set_charsize	setcharsize
set_charspace	setcharspace
set_charup_2	setcharup2
set_charup_3	setcharup3
set_coordinate_system_type	setcoordsystype
set_detectability	setdetectability
set_drag	setdrag
set_echo	setecho
set_echo_group	setechogroup
set_echo_position	setechoposition
set_echo_surface	setechosurface
set_fill_index	setfillindex
set_font	setfont
set_front_plane_clipping	setfrontclip
set_highlighting	sethighlighting
set_image_transformation_2	setimgtransform2
set_image_transformation_3	setimgtransform3
set_image_transformation_type	setimgxformtype
set_image_translate_2	setimgtranslate2
set_image_translate_3	setimgtranslate3
set_keyboard	setkeyboard
set_light_direction	setlightdirect
set_line_index	setlineindex
set_linestyle	setlinestyle
set_linewidth	setlinewidth
set_locator_2	setlocator2
set_marker_symbol	setmarkersymbol
set_ndc_space_2	setndcspace2
set_ndc_space_3	setndcspace3
set_output_clipping	setoutputclip
set_pen	setpen
set_pick	setpick
set_pick_id	setpickid
set_polygon_edge_style	setpolyedgestyle
set_polygon_interior_style	setpolyintrstyle
set_primitive_attributes	setprimattribs
set_projection	setprojection
set_rasterop	setrasterop

Table E-3 *Correspondence Between C Names and Pascal Names— Continued*

C Name	*Pascal Name*
set_segment_detectability	setsegdetectable
set_segment_highlighting	setseghighlight
set_segment_image_transformation_2	setsegimgxform2
set_segment_image_transformation_3	setsegimgxform3
set_segment_image_translate_2	setsegimgxlate2
set_segment_image_translate_3	setsegimgxlate3
set_segment_visibility	setsegvisibility
set_shading_parameters	setshadingparams
set_stroke	setstroke
set_text_index	settextindex
set_valuator	setvaluator
set_vertex_indices	setvertexindices
set_vertex_normals	setvertexnormals
set_view_depth	setviewdepth
set_view_plane_distance	setviewplanedist
set_view_plane_normal	setviewplanenorm
set_view_reference_point	setviewrefpoint
set_view_up_2	setviewup2
set_view_up_3	setviewup3
set_viewing_parameters	setviewingparams
set_viewport_2	setviewport2
set_viewport_3	setviewport3
set_visibility	setvisibility
set_window	setwindow
set_window_clipping	setwindowclip
set_world_coordinate_matrix_2	setworldmatrix2
set_world_coordinate_matrix_3	setworldmatrix3
set_zbuffer_cut	setzbuffercut
size_raster	sizeraster
terminate_core	terminatecore
terminate_device	terminatedevice
terminate_view_surface	terminatevwsurf
text	puttext

E.4. Type Declarations

The list on the following pages is a complete alphabetical list of the Pascal data structures in *SunCore*.

```
type iarr =  array[1..256] of integer;
type parr =  array[1..256] of real;
type cct =  array[1..257] of char;
type ivarray =  array[1..4,1..4] of real;
type ivarray1 =  array[1..3,1..3] of real;
type  pttype = record
       x,y,z,w:real;
    end;
type aspect = record
       width, height:real;
```

```
    end;
type  primattr = record
       lineindx: integer;
       fillindx: integer;
       textindx: integer;
       linestyl: integer;
       polyintstyl: integer;
       polyedgstyl: integer;
       linwidth: real;
       pen: integer;
       font: integer;
       charsize: aspect;
       chrup, chrpath, chrspace: pttype;
       chjust: integer;
       chqualty: integer;
       marker: integer;
       pickid: integer;
       rasterop: integer;
    end;
type rasttyp =  record
            width: integer;
            height: integer;
            depth: integer;
            bits: integer; {var}
            end;
type  cmap = record
            typ: integer;
            nbyt: integer;
            dat :integer; {var}
            end;
type  windtype = record
                 xmin, xmax, ymin, ymax:real;
            end;
type  porttype = record
                 xmin,xmax,ymin,ymax,zmin,zmax:real;
            end;
type vwprmtype = record
                 vwrefpt: array [1..3] of real;
                 vwplnorm: array [1..3] of real;
                 viewdis:real;
                 frontdis:real;
                 backdis:real;
                 projtype:integer;
                 projdir: array [1..3] of real;
                 window:windtype;
                 vwupdir: array [1..3] of real;
                 viewport:porttype;
                  end;
type  vwsurf = record
                 screenname: array [1..DEVNAMESIZE] of char;
                 windowname: array [1..DEVNAMESIZE] of char;
                 windowfd:integer;
                 dd:integer;
```

```
                instance:integer;
                cmapsize:integer;
                cmapname: array [1..DEVNAMESIZE] of char;
                flags:integer;
                ptr: integer;
    end;
type  vwsurfst = array [1..DEVNAMESIZE] of char;
type vwarr =  array[1..MAXVSURF] of vwsurf;
```

E.5. Function Declarations

The list on the following pages is a complete alphabetical list of the Pascal functions in *SunCore*.

```
function allocateraster(var rptr:rasttyp):integer; external;
function awaitanybutton(tim:integer; var buttonnum:integer):integer; external;
function awtbuttongetloc2(time:integer; locatornum:integer;
    var buttonnum:integer; var x:real; var y:real):integer; external;
function awtbuttongetval(time:integer; valnum:integer; var buttonnum:integer;
    var val:real):integer; external;
function awaitkeyboard(tim:integer; keynum:integer; var sptr:cct;
    var length:integer):integer; external;
function awaitpick(time:integer; picknum:integer; var segnam:integer;
    var pickid:integer):integer; external;
function awaitstroke2(tim:integer; picknum:integer; asize:integer; var x:parr;
    var y:parr; numxy:integer):integer; external;
function beginbatchupdate:integer; external;
function closeretainseg:integer; external;
function closetempseg:integer; external;
function createretainseg(segname:integer):integer; external;
function createtempseg:integer; external;
function defcolorindices(var surfacename:vwsurf; i1:integer; i2:integer;
    var r:parr; var g:parr; var b:parr):integer; external;
function delallretainsegs:integer; external;
function delretainsegment(segname:integer):integer; external;
function deselectvwsurf(var surfacename:vwsurf):integer; external;
function endbatchupdate:integer; external;
function filetoraster(var rasfid:text; var rptr:rasttyp; var map:cmap)
    :integer; external;
function freeraster(var rptr:rasttyp):integer; external;
function getmousestate(devclass:integer; devnum:integer; var x:real;
    var y:real; var buttons:integer):integer; external;
function getraster(var surfacename:vwsurf; xmin:real; xmax:real; ymin:real;
    ymax:real; xd:integer; yd:integer; var rptr:rasttyp):integer; external;
function getviewsurface(var surfacename:vwsurf):integer; external;
function initializecore(outputlevel:integer; inputlevel:integer;
    dimension:integer):integer; external;
function initializedevice(deviceclass:integer; devicenum:integer)
    :integer; external;
function initializevwsurf(var surfacename:vwsurf; typ:integer)
    :integer; external;
function inqcharjust(var chjust:integer):integer; external;
function inqcharpath2(var x:real; var y:real):integer; external;
function inqcharpath3(var x:real; var y:real; var z:real):integer; external;
function inqcharprecision(var chquality:integer):integer; external;
```

```
function inqcharsize(var width:real; var height:real):integer; external;
function inqcharspace(var space:real):integer; external;
function inqcharup2(var x:real; var y:real):integer; external;
function inqcharup3(var x:real; var y:real; var z:real):integer; external;
function inqcolorindices(var surfacename:vwsurf; i1:integer; i2:integer;
    var r:parr; var g:parr; var b:parr):integer; external;
function inqcurrpos2(var x:real; var y:real):integer; external;
function inqcurrpos3(var x:real; var y:real; var z:real):integer; external;
function inqdetectability(var detect:integer):integer; external;
function inqecho(devclass:integer; devnum:integer; var echotype:integer)
    :integer; external;
function inqechoposition(devclass:integer; devnum:integer; var x:real;
    var y:real):integer; external;
function inqechosurface(devclass:integer; devnum:integer;
    var surfacename:vwsurf):integer; external;
function inqfillindex(var color:integer):integer; external;
function inqfont(var font:integer):integer; external;
function inqhighlighting(var highlight:integer):integer; external;
function inqimgtransform2(var sx:real; var sy:real; var a:real; var tx:real;
    var ty:real):integer; external;
function inqimgtransform3(var sx:real; var sy:real; var sz:real; var ax:real;
    var ay:real; var az:real; var tx:real; var ty:real; var tz:real)
    :integer; external;
function inqimgxformtype(var segtype:integer):integer; external;
function inqimgtranslate2(var tx:real; var ty:real):integer; external;
function inqimgtranslate3(var tx:real; var ty:real; var tz:real)
    :integer; external;
function inqinvcompmatrix(var iarray:ivarray):integer; external;
function inqkeyboard(keynum:integer; var bufsize:integer; var string:cct;
    var pos:integer):integer; external;
function inqlineindex(var color:integer):integer; external;
function inqlinestyle(var linestyle:integer):integer; external;
function inqlinewidth(var linewidth:real):integer; external;
function inqlocator2(locnum:integer; var x:real; var y:real):integer; external;
function inqmarkersymbol(var mark:integer):integer; external;
function inqndcspace2(var width:real; var height:real):integer; external;
function inqndcspace3(var width:real; var height:real; var depth:real)
    :integer; external;
function inqopenretainseg(var segname:integer):integer; external;
function inqopentempseg(var open:integer):integer; external;
function inqpen(var pen:integer):integer; external;
function inqpickid(var pick:integer):integer; external;
function inqpolyedgestyle(var pestyle:integer):integer; external;
function inqpolyintrstyle(var pistyle:integer):integer; external;
function inqprimattribs(var defprim:primattr):integer; external;
function inqprojection(var ptype:integer; var dx:real; var dy:real;
    var dz:real):integer; external;
function inqrasterop(var rastop:integer):integer; external;
function inqretainsegname(arraycnt:integer; var seglist:iarr;
    var segcnt:integer):integer; external;
function inqretainsegsurf(segname:integer; arraycnt:integer;
    var surflist:vwarr; var surfcnt:integer):integer; external;
function inqsegdetectable(segname:integer; var dtable:integer)
```

```
    :integer; external;
function inqseghighlight(segname:integer; var highlight:integer)
    :integer; external;
function inqsegimgxform2(segname:integer; var sx:real; var sy:real;
    var a:real; var tx:real; var ty:real):integer; external;
function inqsegimgxform3(segname:integer; var sx:real; var sy:real;
    var sz:real; var rx:real; var ry:real; var rz:real; var tx:real;
    var ty:real; var tz:real):integer; external;
function inqsegimgxfrmtyp(segname:integer; var segtype:integer)
    :integer; external;
function inqsegimgxlate2(segname:integer; var tx:real; var ty:real)
    :integer; external;
function inqsegimgxlate3(segname:integer; var sx:real; var sy:real;
    var sz:real):integer; external;
function inqsegvisibility(segname:integer; var visible:integer)
    :integer; external;
function inqstroke(strokenum:integer; var bufsize:integer; var dist:real;
    var time:integer):integer; external;
function inqtextextent2(var string:cct; var dx:real; var dy:real)
    :integer; external;
function inqtextextent3(var string:cct; var dx:real; var dy:real; var dz:real)
    :integer; external;
function inqtextindex(var color:integer):integer; external;
function inqvaluator(valnum:integer; var init:real; var low:real;
    var high:real):integer; external;
function inqviewdepth(var fdist:real; var bdist:real):integer; external;
function inqviewplanedist(var vdist:real):integer; external;
function inqviewplanenorm(var dx:real; var dy:real; var dz:real)
    :integer; external;
function inqviewrefpoint(var rx:real; var ry:real; var rz:real)
    :integer; external;
function inqviewup2(var dx:real; var dy:real):integer; external;
function inqviewup3(var dx:real; var dy:real; var dz:real):integer; external;
function inqvwgcntrlparms(var wclip:integer; var fclip:integer;
    var bclip:integer; var typ:integer):integer; external;
function inqviewingparams(var viewparm:vwprmtype):integer; external;
function inqviewport2(var xmin:real; var xmax:real; var ymin:real;
    var ymax:real):integer; external;
function inqviewport3(var xmin:real; var xmax:real; var ymin:real;
    var ymax:real; var zmin:real; var zmax:real):integer; external;
function inqvisibility(var visible:integer):integer; external;
function inqwindow(var umin:real; var umax:real; var vmin:real;
    var vmax:real):integer; external;
function inqworldmatrix2(var iarray:ivarray1):integer; external;
function inqworldmatrix3(var iarray:ivarray):integer; external;
function lineabs2(x:real; y:real):integer; external;
function lineabs3(x:real; y:real; z:real):integer; external;
function linerel2(x:real; y:real):integer; external;
function linerel3(x:real; y:real; z:real):integer; external;
function mapndctoworld2(ndx:real; ndy:real; var wldx:real; var wldy:real)
    :integer; external;
function mapndctoworld3(ndx:real; ndy:real; ndz:real; var wldx:real;
    var wldy:real; var wldz:real):integer; external;
```

```
function mapworldtondc2(wldx:real; wldy:real; var ndx:real; var ndy:real)
    :integer; external;
function mapworldtondc3(wldx:real; wldy:real; wldz:real; var ndx:real;
    var ndy:real; var ndz:real):integer; external;
function markerabs2(mx:real; my:real):integer; external;
function markerabs3(mx:real; my:real; mz:real):integer; external;
function markerrel2(dx:real; dy:real):integer; external;
function markerrel3(dx:real; dy:real; dz:real):integer; external;
function moveabs2(x:real; y:real):integer; external;
function moveabs3(x:real; y:real; z:real):integer; external;
function moverel2(x:real; y:real):integer; external;
function moverel3(x:real; y:real; z:real):integer; external;
function newframe:integer; external;
function pasloc(function f:integer):integer; external;
function polygonabs2(var xcoor:parr; var ycoor:parr; n:integer)
    :integer; external;
function polygonabs3(var xcoor:parr; var ycoor:parr; var zcoor:parr; n:integer)
    :integer; external;
function polygonrel2(var xcoor:parr; var ycoor:parr; n:integer)
    :integer; external;
function polygonrel3(var xcoor:parr; var ycoor:parr; var zcoor:parr; n:integer)
    :integer; external;
function polylineabs2(var xcoor:parr; var ycoor:parr; n:integer)
    :integer; external;
function polylineabs3(var xcoor:parr; var ycoor:parr; var zcoor:parr;
    n:integer):integer; external;
function polylinerel2(var xcoor:parr; var ycoor:parr; n:integer)
    :integer; external;
function polylinerel3(var xcoor:parr; var ycoor:parr; var zcoor:parr;
    n:integer):integer; external;
function polymarkerabs2(var xcoor:parr; var ycoor:parr; n:integer)
    :integer; external;
function polymarkerabs3(var xcoor:parr; var ycoor:parr; var zcoor:parr;
    n:integer):integer; external;
function polymarkerrel2(var xcoor:parr; var ycoor:parr; n:integer)
    :integer; external;
function polymarkerrel3(var xcoor:parr; var ycoor:parr; var zcoor:parr;
    n:integer):integer; external;
function printerror(var string:cct; error:integer):integer; external;
function putraster(var rptr:rasttyp):integer; external;
function puttext(var string:cct):integer; external;
function rastertofile(var rptr:rasttyp; var map:cmap; var rasfid:text;
    n:integer):integer; external;
function renameretainseg(segname:integer; newname:integer):integer; external;
function reportrecenterr(var error:integer):integer; external;
function restoresegment(segname:integer; var fname:cct):integer; external;
function savesegment(segname:integer; var fname:cct):integer; external;
function selectvwsurf(var surfacename:vwsurf):integer; external;
function setbackclip(onoff:integer):integer; external;
function setcharjust(chjust:integer):integer; external;
function setcharpath2(dx:real; dy:real):integer; external;
function setcharpath3(dx:real; dy:real; dz:real):integer; external;
function setcharprecision(chquality:integer):integer; external;
```

```
function setcharsize(chwid:real; chht:real):integer; external;
function setcharspace(space:real):integer; external;
function setcharup2(dx:real; dy:real):integer; external;
function setcharup3(dx:real; dy:real; dz:real):integer; external;
function setcoordsystype(typ:integer):integer; external;
function setdetectability(detect:integer):integer; external;
function setdrag(drag:integer):integer; external;
function setecho(devclass:integer; devnum:integer; echotype:integer)
    :integer; external;
function setechogroup(devclass:integer; var devarray:iarr; n:integer;
    echotype:integer):integer; external;
function setechoposition(devclass:integer; devnum:integer; x:real; y:real)
    :integer; external;
function setechosurface(devclass:integer; devnum:integer;
    var surfacename:vwsurf):integer; external;
function setfillindex(color:integer):integer; external;
function setfont(font:integer):integer; external;
function setfrontclip(onoff:integer):integer; external;
function sethighlighting(highlight:integer):integer; external;
function setimgtransform2(sx:real; sy:real; a:real; tx:real; ty:real)
    :integer; external;
function setimgtransform3(sx:real; sy:real; sz:real; ax:real; ay:real; az:real;
    tx:real; ty:real; tz:real):integer; external;
function setimgxformtype(segtype:integer):integer; external;
function setimgtranslate2(tx:real; ty:real):integer; external;
function setimgtranslate3(tx:real; ty:real; tz:real):integer; external;
function setkeyboard(keynum:integer; bufsize:integer; var string:cct;
    pos:integer):integer; external;
function setlightdirect(dx:real; dy:real; dz:real):integer; external;
function setlineindex(color:integer):integer; external;
function setlinestyle(style:integer):integer; external;
function setlinewidth(width:real):integer; external;
function setlocator2(locnum:integer; x:real; y:real):integer; external;
function setmarkersymbol(mark:integer):integer; external;
function setndcspace2(width:real; height:real):integer; external;
function setndcspace3(width:real; height:real; depth:real):integer; external;
function setoutputclip(onoff:integer):integer; external;
function setpen(pen:integer):integer; external;
function setpick(pickid:integer; aperture:real):integer; external;
function setpickid(pickid:integer):integer; external;
function setpolyedgestyle(pestyle:integer):integer; external;
function setpolyintrstyle(pistyle:integer):integer; external;
function setprimattribs(var defprim:primattr):integer; external;
function setprojection(ptype:integer; dx:real; dy:real; dz:real)
    :integer; external;
function setrasterop(rop:integer):integer; external;
function setsegdetectable(segname:integer; detectbl:integer):integer; external;
function setseghighlight(segname:integer; highlight:integer):integer; external;
function setsegimgxform2(segname:integer; sx:real; sy:real; a:real; tx:real;
    ty:real):integer; external;
function setsegimgxform3(segname:integer; sx:real; sy:real; sz:real; rx:real;
    ry:real; rz:real; tx:real; ty:real; tz:real):integer; external;
function setsegimgxlate2(segname:integer; tx:real; ty:real):integer; external;
```

```
function setsegimgxlate3(segname:integer; tx:real; ty:real; tz:real)
    :integer; external;
function setsegvisibility(segname:integer; visible:integer):integer; external;
function setshadingparams(amb:real; dif:real; spec:real; flood:real; bump:real;
    hue:integer; style:integer):integer; external;
function setstroke(strokenum:integer; bufsize:integer; dist:real; time:integer)
    :integer; external;
function settextindex(color:integer):integer; external;
function setvaluator(valnum:integer; init:real; low:real; high:real):integer;
    external;
function setvertexindices(var x:iarr; n:integer):integer; external;
function setvertexnormals(var xcoor:parr; var ycoor:parr; var zcoor:parr;
    n:integer):integer; external;
function setviewdepth(near:real; far:real):integer; external;
function setviewplanedist(dist:real):integer; external;
function setviewplanenorm(dx:real; dy:real; dz:real):integer; external;
function setviewrefpoint(x:real; y:real; z:real):integer; external;
function setviewup2(dx:real; dy:real):integer; external;
function setviewup3(dx:real; dy:real; dz:real):integer; external;
function setviewingparams(var viewparm:vwprmtype):integer; external;
function setviewport2(xmin:real; xmax:real; ymin:real; ymax:real)
    :integer; external;
function setviewport3(xmin:real; xmax:real; ymin:real; ymax:real; zmin:real;
    zmax:real):integer; external;
function setvisibility(visibility:integer):integer; external;
function setwindow(umin:real; umax:real; vmin:real; vmax:real)
    :integer; external;
function setwindowclip(onoff:integer):integer; external;
function setworldmatrix2(var iarray:ivarray1):integer; external;
function setworldmatrix3(var iarray:ivarray):integer; external;
function setzbuffercut(var surfacename:vwsurf; var x:parr; var z:parr;
    n:integer):integer; external;
function sizeraster(var surfacename:vwsurf; xmin:real; xmax:real; ymin:real;
    ymax:real; var rptr:rasttyp):integer; external;
function terminatecore:integer; external;
function terminatedevice(devclass:integer; devnum:integer):integer; external;
function terminatevwsurf(var surfacename:vwsurf):integer; external;
```

Note: since *vwarr* is an array of MAXVSURF viewsurfaces, *arraycnt* should be MAXVSURF.

```
function inqsegdetectable(segname:integer;var dtable:integer)
            :integer; external;
function inqseghighlight(segname:integer;var highlight:integer)
            :integer; external;
function inqsegimgxform2(segname:integer;var sx:real;var sy:real;
            var a:real;var tx:real;var ty:real
            ):integer; external;
function inqsegimgxform3(segname:integer;var sx:real;var sy:real;
                var sz:real;var rx:real;var ry:real;
                var rz:real;var tx:real;var ty:real;var tz:real
):integer; external;
function inqsegimgxfrmtyp(segname:integer;var segtype:integer)
            :integer; external;
function inqsegimgxlate2(segname:integer;var tx:real;var ty:real)
```

```
            :integer; external;
function inqsegimgxlate3(segname:integer;var sx:real;var sy:real;
                var sz:real):integer; external;
function inqsegvisibility(segname:integer;var visible:integer):
            integer; external;
function inqstroke(strokenum:integer;var bufsize:integer;var
            dist:real;var time:integer):integer; external;
function inqtextextent2(var string:cct;var dx:real; var dy:real
                ):integer; external;
function inqtextextent3(var string:cct;var dx:real; var dy:real
                ; var dz:real):integer; external;
function inqtextindex(var color:integer):integer; external;
function inqvaluator(valnum:integer;var init:real;var low:real;var high:real)
                :integer; external;
function inqviewdepth(var fdist:real;var bdist:real)
                :integer; external;
function inqviewplanedist(var vdist:real):integer; external;
function inqviewplanenorm(var dx:real; var dy:real;
                var dz:real):integer; external;
function inqviewrefpoint(var rx:real; var ry:real;
                var rz:real):integer; external;
function inqviewup2(var dx:real; var dy:real
                ):integer; external;
function inqviewup3(var dx:real; var dy:real;
                var dz:real):integer; external;
function inqvwgcntrlparms(var wclip:integer;var fclip:integer;
                var bclip:integer;var typ:integer)
                :integer; external;
function inqviewingparams(var viewparm:vwprmtype):integer; external;
function inqviewport2(var xmin:real; var xmax:real;var ymin:real;var ymax:real
                ):integer; external;
function inqviewport3(var xmin:real; var xmax:real;var ymin:real;var ymax:real
                ;var zmin:real;var zmax:real)
                :integer; external;
function inqvisibility(var visible:integer)
            :integer; external;
function inqwindow(var umin:real; var umax:real;var vmin:real;var vmax:real
                ):integer; external;
function inqworldmatrix2(var iarray:ivarray1):integer; external;
function inqworldmatrix3(var iarray:ivarray):integer; external;
function lineabs2(x:real;y:real):integer; external;
function lineabs3(x:real;y:real;z:real):integer; external;
function linerel2(x:real;y:real):integer; external;
function linerel3(x:real;y:real;z:real):integer; external;
function mapndctoworld2(ndx:real; ndy:real;
                var wldx:real; var wldy:real)
                :integer; external;
function mapndctoworld3(ndx:real; ndy:real; ndz:real;
                var wldx:real; var wldy:real
                ; var wldz:real)
                :integer; external;
function mapworldtondc2(wldx:real; wldy:real;
                var ndx:real; var ndy:real)
```

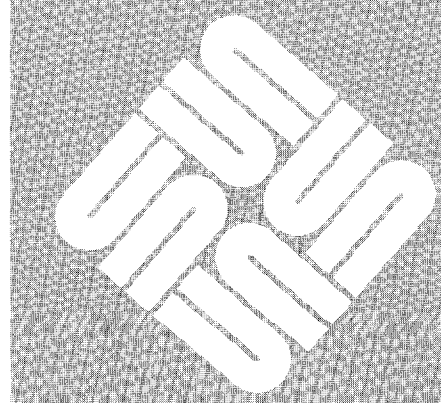
```
                :integer; external;
function mapworldtondc3(wldx:real; wldy:real; wldz:real;
                var ndx:real; var ndy:real
                ; var ndz:real
):integer; external;
function markerabs2(mx:real;my:real):integer; external;
function markerabs3(mx:real; my:real;mz:real):integer; external;
function markerrel2(dx:real;dy:real):integer; external;
function markerrel3(dx:real; dy:real;dz:real):integer; external;
function moveabs2(x:real;y:real):integer; external;
function moveabs3(x:real;y:real;z:real):integer; external;
function moverel2(x:real;y:real):integer; external;
function moverel3(x:real;y:real;z:real):integer; external;
function newframe:integer; external;
function pasloc(function f:integer
            ):integer; external;
function polygonabs2(var xcoor:parr; var ycoor:parr;
        n:integer):integer; external;
function polygonabs3(var xcoor:parr; var ycoor:parr;var zcoor:parr;
        n:integer):integer; external;
function polygonrel2(var xcoor:parr; var ycoor:parr;
        n:integer):integer; external;
function polygonrel3(var xcoor:parr; var ycoor:parr;var zcoor:parr;
        n:integer):integer; external;
function polylineabs2(var xcoor:parr; var ycoor:parr;
        n:integer):integer; external;
function polylineabs3(var xcoor:parr; var ycoor:parr;var zcoor:parr;
        n:integer):integer; external;
function polylinerel2(var xcoor:parr;var ycoor:parr;
        n:integer):integer; external;
function polylinerel3(var xcoor:parr; var ycoor:parr;var zcoor:parr;
        n:integer):integer; external;
function polymarkerabs2(var xcoor:parr; var ycoor:parr;
        n:integer):integer; external;
function polymarkerabs3(var xcoor:parr; var ycoor:parr;var zcoor:parr;
        n:integer):integer; external;
function polymarkerrel2(var xcoor:parr; var ycoor:parr;
        n:integer):integer; external;
function polymarkerrel3(var xcoor:parr; var ycoor:parr;var zcoor:parr;
        n:integer):integer; external;
function printerror(var string:cct;error:integer):integer; external;
function putraster(var rptr:rasttyp):integer; external;
function puttext(var string:cct):integer; external;
function rastertofile(var rptr:rasttyp;var map:cmap;rasfid:integer
                ):integer; external;
function renameretainseg(segname:integer;newname:integer):integer; external;
function reportrecenterr(var error:integer):integer; external;
function restoresegment(segname:integer;var fname:cct):integer; external;
function savesegment(segname:integer;var fname:cct):integer; external;
function selectvwsurf(surfacename:vwsurf
            ):integer; external;
function setbackclip(onoff:integer):integer; external;
function setcharjust(chjust:integer):integer; external;
```

```
function setcharpath2(dx:real; dy:real):integer; external;
function setcharpath3(dx:real; dy:real;dz:real):integer; external;
function setcharprecision(chquality:integer):integer; external;
function setcharsize(chwid:real;chht:real):integer; external;
function setcharspace(space:real):integer; external;
function setcharup2(dx:real; dy:real):integer; external;
function setcharup3(dx:real; dy:real;dz:real):integer; external;
function setcoordsystype(typ:integer):integer; external;
function setdetectability(detect:integer):integer; external;
function setdrag(drag:integer):integer; external;
function setecho(devclass:integer;devnum:integer;
           echotype:integer):integer; external;
function setechogroup(devclass:integer;var devarray:iarr;n:integer;
           echotype:integer):integer; external;
function setechoposition(devclass:integer;devnum:integer;
           x:real;y:real):integer; external;
function setechosurface(devclass:integer;devnum:integer;
        surfacename:vwsurf):integer; external;
function setfillindex(color:integer):integer; external;
function setfont(font:integer):integer; external;
function setfrontclip(onoff:integer):integer; external;
function sethighlighting(highlight:integer):integer; external;
function setimgtransform2(sx:real; sy:real;a:real
               ;tx:real; ty:real):integer; external;
function setimgtransform3(sx:real; sy:real;sz:real;
               ax:real; ay:real;az:real;
               tx:real; ty:real;tz:real)
               :integer; external;
function setimgxformtype(segtype:integer):integer; external;
function setimgtranslate2(tx:real; ty:real):integer; external;
function setimgtranslate3(tx:real; ty:real;tz:real):integer; external;
function setkeyboard(keynum:integer;bufsize:integer;var string:cct;
               pos:integer):integer; external;
function setlightdirect(dx:real; dy:real;dz:real
               ):integer; external;
function setlineindex(color:integer):integer; external;
function setlinestyle(style:integer):integer; external;
function setlinewidth(width:real):integer; external;
function setlocator2(locnum:integer;x:real;y:real):integer; external;
function setmarkersymbol(mark:integer):integer; external;
function setndcspace2(width:real;height:real):integer; external;
function setndcspace3(width:real;height:real;depth:real)
               :integer; external;
function setoutputclip(onoff:integer):integer; external;
function setpen(pen:integer):integer; external;
function setpick(picknum:integer; aperture: real):integer; external;
function setpickid(pickid:integer):integer; external;
function setpolyedgestyle(pestyle:integer):integer; external;
function setpolyintrstyle(pistyle:integer):integer; external;
function setprimattribs(var defprim:primattr):integer; external;
function setprojection(ptype:integer;dx:real; dy:real;dz:real)
               :integer; external;
function setrasterop(rop:integer):integer; external;
```

```
function setsegdetectable(segname:integer; detectbl:integer)
                :integer; external;
function setseghighlight(segname:integer; highlight:integer)
                :integer; external;
function setsegimgxform2(segname:integer;sx:real; sy:real;a:real;
                tx:real;ty:real):integer; external;
function setsegimgxform3(segname:integer; sx:real;  sy:real;
                 sz:real; rx:real;  ry:real; rz:real
                ; tx:real;  ty:real; tz:real
                ):integer; external;
function setsegimgxlate2(segname:integer;tx:real; ty:real
                ):integer; external;
function setsegimgxlate3(segname:integer;tx:real; ty:real;tz:real
                ):integer; external;
function setsegvisibility(segname:integer;visible:integer):integer; external;
function setshadingparams(amb:real;dif:real;spec:real;flood:real;
                bump:real;hue:integer;style:integer
                ):integer; external;
function setstroke(strokenum:integer;bufsize:integer;
            dist:real;time:integer)
            :integer; external;
function settextindex(color:integer):integer; external;
function setvaluator(valnum:integer;init:real;low:real;high:real)
            :integer; external;
function setvertexindices(var x:iarr;n:integer):integer; external;
function setvertexnormals(var xcoor:parr; var ycoor:parr;var zcoor:parr;
        n:integer):integer; external;
function setviewdepth(near:real;far:real):integer; external;
function setviewplanedist(dist:real):integer; external;
function setviewplanenorm(dx:real; dy:real;dz:real):integer; external;
function setviewrefpoint(x:real; y:real;z:real):integer; external;
function setviewup2(dx:real; dy:real):integer; external;
function setviewup3(dx:real; dy:real;dz:real):integer; external;
function setviewingparams(var viewparm:vwprmtype):integer; external;
function setviewport2(xmin:real;xmax:real;ymin:real;ymax:real):
            integer; external;
function setviewport3(xmin:real;xmax:real;ymin:real;ymax:real;zmin:real;zmax:real)
        :integer; external;
function setvisibility(visibility:integer):integer; external;
function setwindow(umin:real;umax:real;vmin:real;vmax:real)
            :integer; external;
function setwindowclip(onoff:integer):integer; external;
function setworldmatrix2(var iarray:ivarray1):integer; external;
function setworldmatrix3(var iarray:ivarray):integer; external;
function setzbuffercut(var surfacename:vwsurf;var x:parr;
    var z:parr;n:integer):integer; external;
function sizeraster(var surfacename:vwsurf;
                xmin:real;xmax:real;ymin:real;ymax:real;
                var rptr:rasttyp):integer; external;
function terminatecore:integer; external;
function terminatedevice(devclass:integer;devnum:integer):integer; external;
function terminatevwsurf(var surfacename:vwsurf):integer; external;
```

F

Hardware Floating Point SunCore Libraries



F

Hardware Floating Point SunCore Libraries

SunCore programs intended for Sun workstations with hardware floating point support may use alternative *SunCore* libraries which provide higher floating point performance. Separate libraries are provided for each of the floating point options described below.

The presence of one of these options is independent of whether a Graphics Processor is present. It is not necessary to use one of these special libraries to take advantage of the Graphics Processor.

For Sun-2 workstations, the only available floating point hardware is the SKY floating point processor. The appropriate library in this case is `/usr/lib/libcoresky.a`. A program linked with this library will only run on a Sun workstation with a SKY board.

For Sun-3 workstations, two floating point hardware options are available. For Sun workstations with the MC68881 floating point co-processor, the appropriate library is `/usr/lib/libcore68881.a`. A program linked with this library will only run on a Sun workstation with an MC68881. For Sun workstations with a Floating Point Accelerator (FPA), the appropriate library is `/usr/lib/libcorefpa.a`. A program linked with this library will only run on a Sun workstation with an FPA.

C programs written with *SunCore* can be compiled with the following command line:

```
% cc -fxxx -o box box.c -lcorexxx -lsunwindow -lpixrect -lm
```

FORTRAN programs written with *SunCore* can be compiled with the following command line:

```
% f77 -fxxx -o box box.f -lcore77 -lcorexxx -lsunwindow -lpixrect -lm
```

Pascal programs written with *SunCore* can be compiled with the following command line:

```
% pc -fxxx -o box box.p -lcorepas -lcorexxx -lsunwindow -lpixrect -lm
```

In these command lines, *xxx* should be replaced with the appropriate symbol

from Table F-1.

Table F-1 *Floating Point Libraries*

Symbol	*Description*
sky	Sky floating point board
68881	MC68881 floating point co-processor
fpa	Floating Point Accelerator

If compiling and linking are done in separate steps, the `-f`*xxx* option must be specified in the linking stage. The `-f`*xxx* option may also be used in the compiling step. Different modules within a program cannot be compiled with different hardware floating point switches, but modules compiled with `-fsoft` or `-fswitch` can be combined with modules compiled with a single type of hardware switch. See the manual pages for `cc`(1), `f77`(1) and `pc`(1) for details.

To compile and link a program to run on any configuration of hardware for a specific processor type (Sun-2 or Sun-3), use the `-fswitch` option for compiling and linking. The `-fswitch` option will cause the compiler to take advantage of floating point hardware if it is available. Otherwise, the compiler will emulate this floating point support with software. See `cc`(1), `f77`(1) or `pc`(1) for details. The `-lcore` option links with the generic *SunCore* library, `/usr/lib/libsuncore.a`. Note that different binary versions of a program are required for Sun-2 and Sun-3 processors.

Many graphics programs written in C do not require the precision implied by evaluating floating point expressions in double precision. The `-fsingle` option may be used to force single precision evaluation of arithmetic expressions involving only `float` quantities (see `cc`(1)).

G

Error Messages

G

Error Messages

SunCore does not use the error numbers suggested by the ACM CORE standard. The following table matches an error number with the error message:

Table G-1 *SunCore Error Messages*

Error Number	*Description*
0	The CORE SYSTEM has already been initialized.
1	The specified level cannot be supported.
2	The surface has already been initialized.
3	No physical surface is associated with the specified logical surface.
4	The CORE SYSTEM has not been initialized.
5	The specified surface has not been initialized.
6	The specified surface is already selected.
7	The specified surface was not selected.
8	A segment is open.
9	The specified surface is not selected.
10	The specified surface has not been deselected.
11	This function has already been called once.
12	A segment has been opened.
13	A value specified for a default attribute is improper.
14	The specified segment does not exist.
15	The VIEW SURFACE ARRAY is not large enough.
16	Segment list overflow, can't create segment.
17	There has been no 'end batch' since last 'begin batch'.
18	There has been no corresponding 'begin batch'.
19	A viewing function has been invoked, or a segment has been created.
20	The value for TYPE is improper.
21	No segment is open.
22	n is <= 0.
23	String contains an illegal character.
24	The vectors established by CHARSPACE and CHARUP are parallel.
25	Invalid marker table offset.
26	Invocation when no open segment.
27	Invalid attribute value.

Table G-1 *SunCore Error Messages— Continued*

Error Number	*Description*
28	Invalid segment type.
29	Invalid segment number.
30	Invalid image transformation for the segment.
31	A retained segment named SEGNAME already exists.
32	The segment type is inconsistent with the current IMAGE_TRANSFORM.
33	No view surface is currently selected.
34	The current viewing specification is inconsistent.
35	No view surfaces have been initialized.
36	There is an existing retained segment named NEW_NAME.
37	There is no retained segment named SEGMENT_NAME.
38	No characters in string (n=0).
39	Dx, dy, and dz, are all zero: no direction can be established.
40	MIN is not less than MAX, for u or v bounds.
41	FRONT_DISTANCE exceeds BACK_DISTANCE; back clip plane is in front.
42	'ndcsp2' or 'ndcsp3' has been invoked since SunCore was last initialized.
43	The invocation of 'ndcspx' is too late, default values have been assumed.
44	A parameter value is greater than 1, or is less than or equal to 0.
45	Neither parameter has a value of 1.
46	Viewport extent is outside of normalized device coordinate space.
47	MIN is not less than MAX, for x, y, or z bounds.
48	Specified device already enabled.
49	DEVICE_CLASS or DEVICE_NUM invalid.
50	DEVICE_CLASS invalid.
51	Specified device is not enabled.
52	LOCATOR_NUM is invalid.
53	The specified LOCATOR device is not enabled.
54	VALUATOR_NUM is invalid.
55	The specified VALUATOR device is not enabled.
56	The TIME value is less than zero.
57	EVENT_CLASS and EVENT_NUM do not specify a valid event device.
58	EVENT_CLASS is not a legal event device class.
59	The specified association already exists.
60	EVENT_CLASS or SAMPLED_CLASS reference invalid or wrong type of class.
61	EVENT_NUM or SAMPLED_NUM are invalid device numbers for their classes.
62	The specified association does not exists.
63	The current event report is not from a PICK device.
64	The current event report is not from a KEYBOARD event.

Table G-1 *SunCore Error Messages— Continued*

Error Number	*Description*
65	Input string was not large enough to hold the string centered by user.
66	When event occurred, the LOCATOR device was not enabled or was not associated with the event device.
67	When event occurred, the VALUATOR device was not enabled or was not associated with the event device.
68	XECHO and YECHO specify positions outside NDC space.
69	PICK_NUM does not specify a valid PICK device.
70	LOCATOR_NUM does not specify a valid LOCATOR device.
71	XLOC,YLOC specify a position outside normalized device coordinate space.
72	VALUATOR_NUM is not a valid VALUATOR device.
73	LOW_VALUE is greater than HIGH_VLAUE.
74	INITIAL_VALUE lies outside the range defined by LOW_VALUE and HIGH_VALUE.
75	KEYBOARD_NUM is not a valid KEYBOARD device.
76	BUFFER_SIZE is <= zero or > the defined maximum.
77	BUTTON_NUM is not a valid BUTTON device.
78	Incorrect arguments for the specified function.
79	Incorrect argument count for the specified function.
80	Specified function not supported.
81	More than MAXPOLY vertices in polygon.
82	Invalid Viewing Specification. Viewing Matrix Unchanged!
83	Invalid view surface name.
84	Selected view surface cannot support hidden surfaces.
85	No other view surface can be initialized at this time.
86	Raster depth is 1 or 8 bit pixels only.
87	Unable to allocate space for virtual memory display list.
88	Memory allocation failure.
89	Error in view reference point.
90	Error in view plane normal.
91	Error in view plane distance.
92	Error in view depth.
93	Error in projection.
94	Error in window.
95	Error in view up direction.
96	Error in viewport.
97	Set_ndc_space_2 or set_ndc_space_3 has already been invoked.
98	The default NDC space has already been established.
99	A parameter is not in the range of 0 to 1.
100	Neither width nor height has a value of 1.
101	Width or height is 0.
102	STROKE_NUM is not a valid STROKE device.
103	Input device is already initialized.
104	Input device is not initialized.

Table G-1 *SunCore Error Messages—Continued*

Error Number	*Description*
105	DEVICE_CLASS is not a valid device class.
106	Invalid echo type for PICK device.
107	Invalid echo type for KEYBOARD device.
108	Invalid echo type for STROKE device.
109	Invalid echo type for LOCATOR device.
110	Invalid echo type for VALUATOR device.
111	Invalid echo type for BUTTON device.
112	Echo position specified is outside NDC space.
113	No BUTTON device is initialized.
114	Invalid raster type.
115	Fewer than 3 vertices in polygon.

H

Type and Structure Definitions

H

Type and Structure Definitions

This appendix lists the types and structures used by *SunCore* functions. The definition of these types and structures can be found in <usercore.h>.

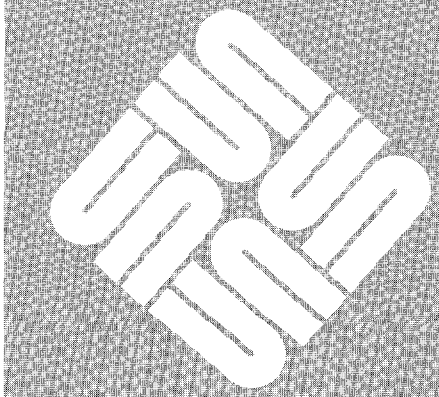
```
#define BASIC        0    /* Core output levels */
#define BUFFERED     1
#define BUTTON       2
#define CENTER       2
#define CHARACTER    1
#define CMR      4
#define CMRBOLD      5
#define COMPLETE     2
#define CONSTANT     0    /* polygon shading modes */
#define DASHED       2
#define DEFAULT_VWSURF(ddname) {"", "", 0, ddname, 0, 0, "", 0, 0}
#define DEVNAMESIZE 20
#define DOTDASHED    3
#define DOTTED       1
#define DYNAMICA     2
#define DYNAMICB     3
#define DYNAMICC     4
#define FALSE        0
#define GACHA        1
#define GACHABOLD    3
#define GALLANT      0    /* raster font constants */
#define GOURAUD      1
#define GREEK        1
#define KEYBOARD     1
#define LEFT         1
#define LOCATOR      3
#define MAXVSURF     5    /* view surfaces; maximum number of */
#define NOINPUT      0    /* Core input levels */
#define NONE         1    /* segment types */
#define NORMAL       0    /* rasterop selection */
#define NULL_VWSURF {"", "", 0, 0, 0, 0, "", 0, 0}
#define OFF      0    /* char justify constants */
#define OLDENGLISH   3
#define ORROP        2
#define PARALLEL     0    /* transform constants */
#define PERSPECTIVE  1
#define PHONG        2
```

```
#define PICK          0   /* input device constants */
#define PLAIN         0   /* polygon interior style */
#define RIGHT         3
#define ROMAN         0   /* vector font select constants */
#define SAIL          2
#define SCRIPT        2
#define SHADED        1
#define SOLID         0   /* line styles */
#define STICK         4
#define STRING        0
#define STROKE        5
#define SYMBOLS       5
#define SYNCHRONOUS   1
#define THREED        1
#define TRUE          1
#define TWOD          0   /* Core dimensions */
#define VALUATOR      4
#define VWSURF_NEWFLG   1
#define XFORM2        3
#define XFORM3        3
#define XLATE2        2
#define XLATE3        2
#define XORROP        1

static struct {      /* default primitive attributes */
    int lineindx;
    int fillindx;
    int textindx;
    int linestyl;
    int polyintstyl;
    int polyedgstyl;
    float linwidth;
    int pen;
    int font;
    float chwidth,chheight;
    float chup[4], chpath[4], chspace[4];
    int chjust;
    int chqualty;
    int marker;
    int pickid;
    int rasterop;
} PRIMATTS = {1,1,1,SOLID,PLAIN,SOLID,0.0,0,STICK,11.,11.,
        {0.,1.,0.,1.},{1.,0.,0.,1.}, {0.,0.,0.,1.},
        OFF,STRING,42,0,NORMAL};

struct vwsurf   {
    char screenname[DEVNAMESIZE];
    char windowname[DEVNAMESIZE];
    int windowfd;
    int (*dd)();
    int instance;
    int cmapsize;
    char cmapname[DEVNAMESIZE];
```

```
    int flags;
    char **ptr;
};
```

I

Example Program



I

Example Program

This appendix contains an example program that uses a number of *SunCore*'s facilities. The example is called *factory*. It displays a factory building with a smokestack and a cloud of smoke puffing out. Silicon chips move in at one end of the building, and Sun Workstations come out of the other end.

Facilities displayed by this simple example include texturing, translation, scaling, and output clipping. The example is presented function by function, with an accompanying narrative.

I.1. Declarations and the Main Program

The first line in a *SunCore* application program should include the file `<usercore.h>` which contains the definitions required for using the *SunCore* graphics package. The `factory` program also has some definitions stored in the file `factory.h`.

Figure I-1 `factory.h` *Header File*

```
#define FACTORY 10
#define CLOUD   9
#define WORKSTATION_1   1
#define WORKSTATION_2   2
#define WORKSTATION_3   3
#define CHIP_1  4
#define CHIP_2  5
#define CHIP_3  6
```

Then there are some definitions. Then we define and initialize the variables that describe the outlines of the various objects in the picture: Then we have the main program: The first call in the program is to initialize *SunCore*, with an appropriate exit if there is an error returned: Then we initialize and select a view surface. Again, we exit if there was an error returned: Then we establish a viewport and a window. Note that we can set clipping on output — this is a *SunCore* extension to the ACM Core. Set up the color lookup table. Now make a temporary segment for a title and border. Next we establish a segment for the factory. This segment is the simplest type, since we perform no transformations of any kind on it. Next we establish a segment for the cloud above the factory. This segment is subject to scaling, so we must allow for transformations. Lastly, we establish segments for the chips and the workstations. The chips and workstations will be moving across the picture, so these segments must allow translation. Notice that

we created the workstations all on top of each other, and also all the chips on top of each other. The actual spatial separation of the individual segments is handled in the main body of the animation code.

Now we get to the body of the code which animates the picture. The outer for loop is done 100 times. The calls on the translation functions make the chips and workstations move. The inner for loop makes the cloud grow: Finally, when everything is done, we deselect the view surface, and terminate *SunCore*: The remainder of the demonstration program consists of the functions which fill in the details in the individual segments.

Figure I-2 `main.c` *Function*

```
#include <usercore.h>
#include "factory.h"

static float delta[] = {0.0, 0.025, 2*0.025, 3*0.025, 4*0.025,
                        5*0.025, 6*0.025, 7*0.025, 8*0.025, 9*0.025,
                        10*0.025, 11*0.025, 12*0.025};
int pixwindd(); /* device driver name for SunWindows */
                /* on a monochrome display - see Appendix B */
struct vwsurf vsurf = DEFAULT_VWSURF(pixwindd);
                /* The DEFAULT_VWSURF macro */
                /* is defined in <usercore.h> */

main()
{
    short i, p0, p1, p2;
    float clx, cly, scale;

    if (initialize_core(DYNAMICB, NOINPUT, TWOD))
        exit(0);
    if (initialize_view_surface(&vsurf, FALSE))
        exit(1);
    if (select_view_surface(&vsurf))
        exit(1);
    set_viewport_2(0.05, 0.95, 0.05, 0.7);
    set_window(30.0, 225.0, 30.0, 225.0);
    set_output_clipping(TRUE);
    set_window_clipping(FALSE);
    create_temporary_segment();
    move_abs_2(30.0, 30.0);
    line_rel_2(0.0, 195.0);
    line_rel_2(195.0, 0.0);
    line_rel_2(0.0, -195.0);
    line_rel_2(-195.0, 0.0);
    set_charprecision(CHARACTER);
    set_charsize(14.0, 14.0);
    set_text_index(1);
    move_abs_2(40.0, 200.0);
    text("SunCore");
    close_temporary_segment();
```

```
set_image_transformation_type(NONE);
create_retained_segment(FACTORY);
factory(110.0, 60.0);
close_retained_segment();
set_image_transformation_type(XFORM2);
create_retained_segment(CLOUD);
map_world_to_ndc_2(120.0, 100.0, &clx, &cly);
set_segment_image_transformation_2(CLOUD, 0.05, 0.1,
    0.0, clx, cly + 0.02);
cloud(0.0, 0.0);
close_retained_segment();
set_image_transformation_type(XLATE2);
    /* Draw the Sun Workstation Segment */
create_retained_segment(WORKSTATION_1);
sunws(160.0, 60.0);
close_retained_segment();
create_retained_segment(WORKSTATION_2);
sunws(160.0, 60.0);
close_retained_segment();
create_retained_segment(WORKSTATION_3);
sunws(160.0, 60.0);
close_retained_segment();
    /* Draw the Chip Segment */
create_retained_segment(CHIP_1);
chip(20.0, 70.0);
close_retained_segment();
create_retained_segment(CHIP_2);
chip(20.0, 70.0);
close_retained_segment();
create_retained_segment(CHIP_3);
chip(20.0, 70.0);
close_retained_segment();
p0 = 0;
p1 = 4;
p2 = 8;
for (i=0; i<100; i++) {
    set_segment_image_translate_2(WORKSTATION_1, delta[p0], 0.0);
    set_segment_image_translate_2(WORKSTATION_2, delta[p1], 0.0);
    set_segment_image_translate_2(WORKSTATION_3, delta[p2], 0.0);
    set_segment_image_translate_2(CHIP_3, delta[p2], 0.0);
    set_segment_image_translate_2(CHIP_2, delta[p1], 0.0);
    set_segment_image_translate_2(CHIP_1, delta[p0], 0.0);
    p0++;
    p1++;
    p2++;
    if (p0 > 11)
        p0 = 0;
    if (p1 > 11)
        p1 = 0;
    if (p2 > 11)
        p2 = 0;
    for (scale=0.1; scale<1.0; scale += 0.2)
        set_segment_image_transformation_2(CLOUD,
```

```
                0.5 * scale, scale, 0.0,
                clx, cly + scale * 0.2);
    }
    deselect_view_surface(&vsurf);
    terminate_core();
}
```

I.2. The Factory Drawing Function

First, here are the coordinates for the outline of the factory itself: The next set of declarations describe the outline of the windows in the factory: Now we have the actual code of the factory drawing function itself: The x0 and y0 arguments to the factory function describe the absolute position in world coordinates at which the factory should appear. The actual outline of the factory is described by the array of coordinates declared above. Now we draw the windows within the factory: The next function is the one which draws the Sun Workstations within the workstation segment.

Figure I-3 `factory.c` *Function*

```
#include <usercore.h>
#include "factory.h"

static float factdx[] = {0.0, 0.0, 8.0, 2.0, 3.0, 2.0, 3.0,
                         1.0, 3.0, 1.0, 17.0, 0.0, -40.0};
static float factdy[] = {0.0, 20.0, 0.0, 20.0, 0.0, -20.0,
                         0.0, 15.0, 0.0, -15.0, 0.0, -20.0, 0.0};
static float winddx[] = {0.0, 0.0, 10.0, 0.0, -10.0};
static float winddy[] = {0.0, 5.0, 0.0, -5.0, 0.0};
static int black = 3;
static int brick = 1;

factory(x0, y0)
float x0, y0;
{
    set_fill_index(brick);
    move_abs_2(x0, y0); /* Move to appropriate position */
    polygon_rel_2(factdx, factdy, 12);  /* Draw the factory outline */
    set_fill_index(black);
    move_rel_2(5.0, 10.0);  /* Move to position of first window */
    polygon_rel_2(winddx, winddy, 4);   /* and draw the window */
    move_rel_2(15.0, 0.0);  /* Move to position of second window */
    polygon_rel_2(winddx, winddy, 4);   /* and draw the window */
    set_fill_index(1);  /* reset fill index */
}
```

I.3. The Workstation Drawing Function

The declarations below describe the outline of the Sun Workstation. Tube describes the screen, Case describes the outer outline of the case, base describes the base of the Workstation, and keybd describes the appearance of the keyboard: Then all we have to do is move to the coordinates that were supplied as function arguments, and draw the lines:

Figure I-4 `sunws.c` *Function*

```
#include <usercore.h>
#include "factory.h"

static float tubex[] = {0.0, 5.0, 0.0, -5.0};
static float tubey[] = {5.0, 0.0, -5.0, 0.0};
static float casex[] = {1.0, 7.0, 1.0, 1.0, -1.0, -7.0, -1.0};
static float casey[] = {7.0, 0.0, -7.0, 1.0, 7.0, 0.0, -1.0};
static float basex[] = {9.0, -1.0, -1.0, -5.0, -1.0};
static float basey[] = {0.0, 0.0, -2.0, 0.0, 2.0};
static float keybdx[] = {0.0, 10.0, 3.0, 0.0, -10.0, -3.0, 10.0, 3.0};
static float keybdy[] = {-1.0, 0.0, 2.0, 2.0, 0.0, -3.0, 0.0, 3.0};

sunws(x0, y0)
float x0, y0;
{
    move_abs_2(x0+5.0, y0+8.0); /* Move to the position given */
    polyline_rel_2(tubex, tubey, 4);    /* Draw the tube */

    move_rel_2(-2.0, -1.0);
    polyline_rel_2(casex, casey, 7);    /* Draw the case */

    move_rel_2(-1.0, -7.0);
    polyline_rel_2(basex, basey, 5);    /* Draw the base */

    move_abs_2(x0, y0+1.0);
    polyline_rel_2(keybdx, keybdy, 8);  /* Draw the keyboard */
}
```

I.4. The Chip Drawing Function

The declarations below describe the outline of the chips. Plasti describes the outline of the chip itself, while lead describes the outline of the leads on the chip: Then all we have to do is move to the coordinates that were supplied as function arguments, and draw the lines:

Figure I-5 `chip.c` *Function*

```
#include <usercore.h>
#include "factory.h"

static float plastix[] = {0.0, 16.0, 0.0, -16.0};
static float plastiy[] = {4.0, 0.0, -4.0, 0.0};

static float leadx[] = {-1.0, 2.0, -1.0, 0.0};
static float leady[] = {2.0, 0.0, -2.0, -4.0};

chip(x0, y0)
float x0, y0;
{
    short i;

    set_rasterop(XORROP);
    move_abs_2(x0, y0); /* Move to appropriate position */
    polyline_rel_2(plastix, plastiy, 4);    /* Draw the chip */
    move_rel_2(2.0, 1.0);

    for (i=0; i<5; i++) {   /* Draw the leads on the chip */
        polyline_rel_2(leadx, leady, 4);
        move_rel_2(3.0, 4.0);
    }

    set_rasterop(NORMAL);   /* Reset rasterop */
}
```

I.5. The Cloud Drawing Function

The last function is the one that draws the cloud. The cloud function is easy: all we have to do is draw its outline. The actual scaling of the cloud is done in the main program.

The declarations below describe the outline of the cloud: Then all we have to do is move to the coordinates that were supplied as function arguments, and draw the lines:

Figure I-6 `cloud.c` *Function*

```
#include <usercore.h>
#include "factory.h"

static float cloudx[] = {0.0, 8.0, -8.0, -4.0, 2.0, 14.0, 8.0, 0.0,
                         12.0, 8.0, 4.0, 0.0, -10.0, 10.0, 4.0, -2.0,
                         -6.0, -12.0, -6.0, -12.0, -10.0};
static float cloudy[] = {12.0, 8.0, 2.0, 6.0, 6.0, 10.0, -4.0, -6.0,
                         10.0, 0.0, -4.0, -10.0, -10.0, -2.0, -6.0,
                         -8.0, -4.0, 0.0, 4.0, -8.0, 4.0};

cloud(x0, y0)
float x0, y0;
{
    move_abs_2(x0, y0);
    polyline_rel_2(cloudx, cloudy, 21);
}
```

Index

T

U

V

W