Table of Contents

Table of Contents
(Continued)

Section Page

List of Tables

List of Examples

1. Using This Manual

The purpose of this manual is to describe the Raster Technologies Display List Firmware, optional with the Model One/80. It is intended for programmers doing applications software development who are familiar with the basic Raster Technologies firmware command set.

The Raster Technologies Model One/80 Display List Firmware Reference manaul includes two sections and an appendix. Section 1 briefly discusses this manual's purpose. Section 2 provides an in-depth overview of the Model One/80 Display List Firmware. This overview describes display list structures, explaining the process of creating, modifying, and manipulating them.

Last, an appendix compares the earlier Model One/25, Model One/40, Model One/60 Display List Firmware with the new Model One/10, Model One/80 Display List Firmware Product. The process of using display list structures with the original commands is first explained in terms of the newer commands. Then a command-by-command comparison is presented.

2. Overview of the Model One/80 Display List Firmware

The Raster Technologies Display List Firmware is optional with the Model One/80. This firmware is a general purpose development tool, well suited to the following applications: electrical CAD, both VLSI chip design and PCB layout; mechanical CAD; cartography and seismic applications; and presentation graphics applications.

2.1 Display List Structures

The basic display list structure is the segment. Segments are lists of commands. The commands comprising segments can be categorized by the user in several ways. They can be grouped together into nested segments or labeled using pick IDs.

Both whole segments and segment parts have attributes. These attributes allow segments and segment subdivisions to be grouped and regrouped into many simultaneous, perhaps-overlapping categories. This process of easily manipulating geometry into groups in order to visually differentiate these groups is one of the basic purposes of display lists. The other is to facilitate rapid redisplay of images when the communications link would otherwise be a limiting factor.

2.1.1 Segments: Characteristics and Storage

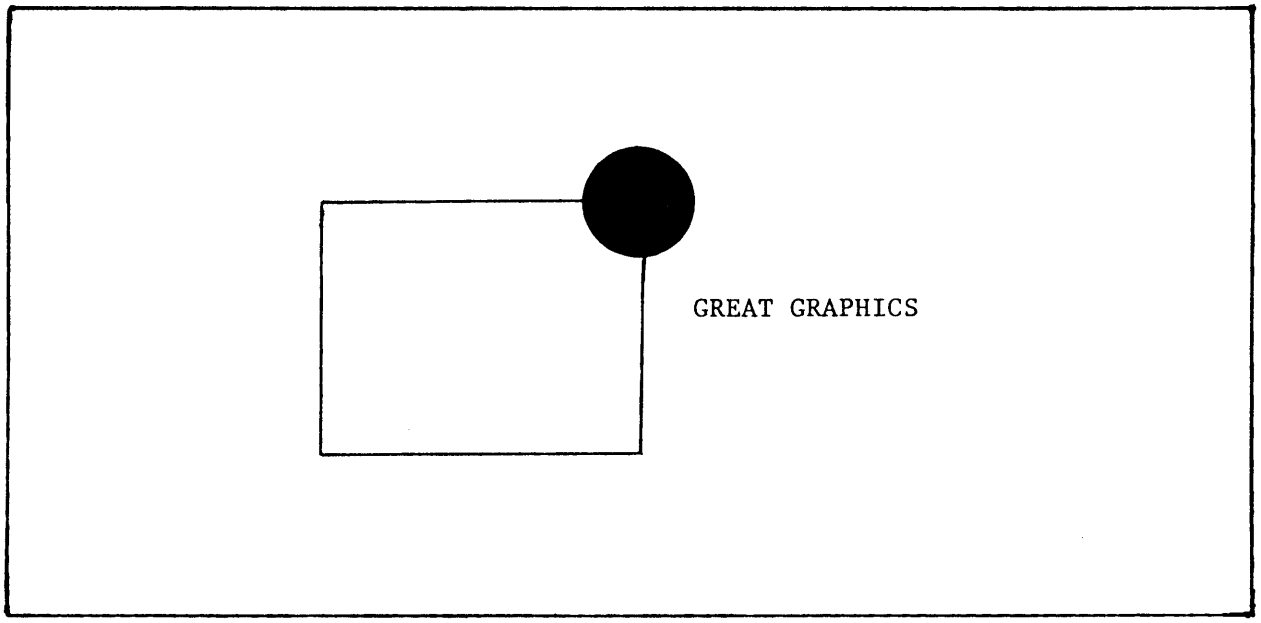
In a display list, graphics primitives are stored and referenced in structures called segments. A segment is a collection of Model One commands, manipulated as a single unit. The following example shows a typical segment definition.

Example 2.1 Typical Segment Definition

```
! SEGINI 0 128        ;Initialize display lists.
! SEGDEF 1            ;Begin definition of Segment 1.
$ VAL8 63             ; Definition begins with setting the current
                        value to white...
$ MOVABS 0 0          ; Move the current point to 0,0...
$ RECTAN 100 80       ; Draw a white rectangle...
$ PRMFIL ON           ; Set PRMFIL to ON; fill subsequent
                        primitives...
$ VAL8 3              ; Set the current value to blue...
$ MOVABS 100 80       ; Move the current point to 100,80...
$ CIRCLE 20           ; Draw a blue circle with radius 20...
$ VAL8 48             ; Set current value to red...
$ MOVABS 120 40       ; Move the current point to 120,40...
$ TEXT1 GREAT GRAPHICS
                      ; Draw text string, "GREAT GRAPHICS"...
$ SEGEND              ;End definition of Segment 1.
!
```

Note that defining Segment 1 does not draw it on your screen. To draw segments (see Section 2.4.1.), execute them with the SEGREF command. When Segment 1 above is drawn, it looks like this:

Example 2.1 (continued)



Each segment is identified by a unique 32-bit integer called the segment ID (Segment 1 in the above example). Segment IDs greater than or equal to FC000000 (hex) are reserved. Both the number of segments and the size of each one are limited only by the amount of display list memory in the system.

On the Model One/80, local display lists are stored on the optional Bulk Memory Card, which is available in multiples of 1 MByte. To facilitate memory management, the firmware divides segments into smaller, fixed-length chunks, called blocks.

You must specify this block size with the SEGINI command. It is important to specify a size that is large enough to allow optimum performance (minimize memory management) but small enough to avoid wasting RAM. The minimum block size is 32 bytes and the maximum is 16,384 bytes. Note that no segment command can be executed until SEGINI is issued.

If you want to check on system storage, the SYSTAT command returns the Display List Firmware memory usage and availability.

2.1.2 Pick IDs

Segment-definition commands can be labeled (delimited) with pick identification numbers (pick IDs). A pick ID is a 32-bit-integer label used to associate a group of graphics primitives with each other and to separate one group from another group.

The PICKID command is a tag, or delimiter, which defines a new pickable entity. This "entity" consists of all subsequent Model One commands until the next PICKID command. Pickable entities can be deleted as a single unit. Pick IDs also facilitate local picking and highlighting (see Section 2.5), in addition to segment editing.

You can also use the INCPID command to define a new pickable entity. The INCPID command increments the current pick ID by one.

In the following example, the same segment definition used in Example 2.1 is subdivided by pick ID labels. Each group of commands that sets the color value, moves the current point, and draws primitives becomes one pickable entity.

Example 2.2 Typical Segment Definition Subdivided by Pick ID Labels

```
! SEGINI 0 128        ;Initialize display lists.
! SEGDEF 1            ;Begin definition of Segment 1.
$ PICKID 1            ; Definition begins with assigning pick
                        identification number 1 to the following
                        commands...
$ VAL8 63             ; Set the current value to white...
$ MOVABS 0 0          ; Move the current point to 0,0...
$ RECTAN 100 80       ; Draw a white rectangle...
$ PICKID 2            ; End Pick ID 1 and begin Pick ID 2; assign
                        Pick ID 2 to the following commands...
$ PRMFIL ON           ; Set PRMFIL to ON; fill subsequent
                        primitives...
$ VAL8 3              ; Set the current value to blue...
$ MOVABS 100 80       ; Move the current point to 100,80...
$ CIRCLE 20           ; Draw a blue circle with radius 20...
$ PICKID 3            ; End Pick ID 2 and begin Pick ID 3; assign
                        Pick ID 3 to the following commands...
$ VAL8 48             ; Set current value to red...
$ MOVABS 120 40       ; Move the current point to 120,40...
$ TEXT1 GREAT GRAPHICS
                      ; Draw text string, "GREAT GRAPHICS"...
$ SEGEND              ;End definition of Segment 1.
!
```

When Segment 1 is now drawn, it looks exactly the same as in Example 2.1. But editing this segment is now possible. For example, the graphics labeled PICKID 1 can be deleted as one entity. To do this, use the DELPID command.

See Section 2.2.2 for more information on segment editing and Section 2.5 for a detailed discussion of picking.

2.1.3 Segment Attributes: Visibility and Pickability

When you define a segment, two attributes, visibility and pickability, are initially set to ON. If a segment is visible, it is drawn when it is executed. On the other hand, if a segment is invisible and executed, the commands in its definition are interpreted (for example, DRAW commands move the current point) but image memory is not altered. Likewise, if a segment is pickable, its commands are subject to being picked during a pick operation.

You can change the visibility and pickability attributes of any segment with the SETATR command.

2.1.4 Segment Identity Information

The Model One stores and returns to you when requested several types of information on segment identity: segment ID numbers, pick ID numbers, and nesting levels. To save this information, the system uses registers, counters, and a pick buffer. To return this information, the Display List Firmware has several readback commands.

2.1.4.1 Registers

Picking operations (see Section 2.5) use two local registers: the segment ID register and the pick ID register. SEGREG contains the current segment ID and PIDREG holds the current pick ID.

The purpose of these registers is to determine what to highlight/unhighlight or what to delete with DELPID. Three commands can update these registers: SEGREF in pick mode (picking) if there is a pick hit, RDPICK, and SETGL.

The readback command RDREG returns the segment ID and the pick ID currently in SEGREG and PIDREG respectively. These registers are used primarily for highlighting but also for deleting pick IDs with the DELPID command.

2.1.4.2 Counters and Pick Buffer

The display list counters and the pick buffer are used, again, for picking operations. These two components of the Model One are discussed in detail in Section 2.5.

2.1.4.3 Reading Back Segment Information

The Display List Firmware readback commands are: RDPICK, RDPID, RDREG, RDXFORM, SEGINQ, and SYSTAT. For all these commands, the format of the readback (binary or ASCII) is determined by the value you previously specified with the RDMODE command.

2.2 Creating and Modifying Segments

After segments have been defined, they can be modified by adding commands to them, deleting commands from them, copying them, renaming them, and deleting them entirely.

2.2.1 Defining Segments

The SEGINI command initializes all display list structures and should, therefore, be used whenever an entirely new set of segments is to be defined. Only after you specify the block size and initialize display list structures are you ready to define segments and assign segment attributes.

The SEGDEF command begins the definition of a segment and sets the visibility and pickabiity attributes to ON for that segment. A segment definition can include a variety of commands, including

- o Commands that draw 2-D graphics primitives.
- o Commands that move the current point.
- o Commands that change the current color.
- o Commands that change primitive-generation attributes, such as VECPAT and PRMFIL.
- o The SEGREF command to nest (reference) child segments.
- o The PICKID command.

You should not put readback commands (e.g., READBU, READP, etc.) in segments. Readback commands in a display list segment will hang the system when the segment is referenced.

You must end every segment definition with the SEGEND command to close the segment. SEGEND also exits the display controller from the sublevel "segment definition" mode, returning it to regular Graphics mode.

Several readback commands give information on defined segments. A list of defined segments can be returned with execution of the SYSTAT command. In addition, the existence of a segment can be determined with the SEGINQ command.

2.2.2 Modifying Segments

Existing segment definitions can be modified in several ways. If you wish, you can add new commands. The SEGAPP command reopens a closed segment so that commands can be appended to it. You must use a SEGEND command to terminate each append.

Replacing portions of a segment definition can be done with a combination of deleting and appending. First, use the DELPID command, previously mentioned in Section 2.1.2, to delete the commands assigned a particular pick ID label. Second, use the SEGAPP command.

You can also delete entire segments with the SEGDEL command, rename segments with the SEGREN command, and copy segments with the SEGCOP command.

2.3 Executing and Nesting Segments

While you are defining and modifying segments, no graphics are generated. To draw defined segments, you must execute, or "SEGREF," them.

2.3.1 Executing Segments

When you are ready to display your segment structures, execute them. The SEGREF command executes the commands within a defined segment. Unless you specify otherwise (see Section 2.4), the geometry is now drawn.

2.3.2 Nesting Segments

You can nest segment definitions to a depth of 16. With nesting, the outer, or primary, segment is called the parent and the segments it references are known as its children.

You can easily nest segments within other segments. Segment nesting is done by embedding SEGREF commands within a segment definition. These SEGREFs can reference both segments that have already been defined and segments not yet defined. If you draw a parent that references as-yet-undefined children, the displayed parent will not show those children segments and no message is generated to alert you.

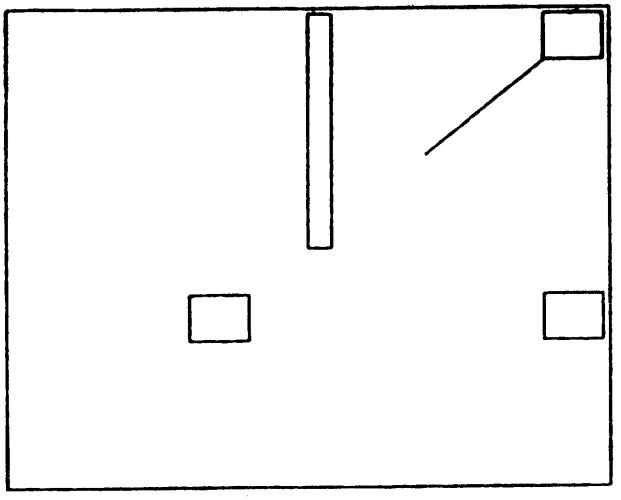
Example 2.3 shows nesting by embedding multiple references to a child, Segment 3, within a parent, Segment 2. Note that child Segment 3 is referenced before it is defined. This is no problem because Segment 3 is defined before its parent is actually executed. However, if Parent Segment 2 were to be executed without nested Segment 3 having been defined, only a rectangle would be drawn.

Example 2.3 Nesting: Multiple References to a Second Segment From Within a Parent Segment

```
! SEGINI 0 128        ;Initialize display lists.
! SEGDEF 2            ;Begin definition of Segment 2.
$ RECREL 20 250       ; Definition begins with a rectangle...
$ MOVABS 100 100      ; followed by a MOVE...
$ DRWABS 200 200      ; followed by a line...
$ SEGREF 3            ; followed by as-yet-undefined Segment 3 (which
                        will be defined as four DRAWs making a
                        square)...
$ MOVABS 200 -100     ; followed by another MOVE...
$ SEGREF 3            ; followed again by Segment 3, a second square...
$ MOVABS -100 -100    ; followed by another MOVE...
$ SEGREF 3            ; followed again by Segment 3, a third square.
$ SEGEND              ;End the definition of Segment 2.

! SEGDEF 3            ;Begin definition of Segment 3.
$ DRWREL 0 50         ; Definition begins with a DRAW...
$ DRWREL 50 0         ; followed by a second DRAW...
$ DRWREL 0 -50        ; followed by a third DRAW...
$ DRWREL -50 0        ; followed by a fourth DRAW, making a square.
$ SEGEND              ;End the definition of Segment 3.

! SEGREF 2            ;Execute (draw) Segment 2.
```



There is no limit to the number of segment references at the same level. For example, Segment 3 (the square) can reference any number of other segments. If one of those segments, in turn, references another segment, the nesting level is three; it is the nesting level that is limited to 16.

Whenever a parent is invisible, so are its children. This is true even if the visibility attribute of any of those children is ON. However, if a parent is visible, any of its children can be made invisible by setting their visibility attribute to OFF with the SETATR command.

A child, likewise, always takes on the pick ID of its parent until a PICKID command within the child alters it. When a nested segment completes execution, the PICKID of the parent returns to its own value at the time of its invocation.

2.4 Execution Modes

With execution modes, you set the display controller for upcoming drawing, picking, highlighting, and unhighlighting operations. Execution modes are set using the EXMODE command. Use this command with the appropriate option before executing or referencing segments to perform the operation you want. EXMODE's options are:

- Normal draw mode (the mode assumed until now during this discussion)
- Pick mode
- Highlight mode
- Unhighlight mode

2.4.1 Normal Draw Mode

Use this mode to draw graphics primitives, to execute other kinds of commands within segments (such as loading a register), and to execute macros. The default of the display controller is EXMODE NORMAL. A COLDstart or WARMstart restores the controller to this state.

2.4.2 Pick Mode

Execute EXMODE PICK before picking is to be performed (see Example 2.3). Picking is described in Section 2.5.

2.4.3 Highlight Mode

Use this mode to locally highlight graphics elements from within segments. Typically, local highlighting is done after picking to confirm the pick. First specify a highlight color by loading it into VREG 45. Then use EXMODE HILITE before drawing primitives in the highlight color (see Example 2.4).

The contents of the segment ID register (SEGREG) and the pick ID register (PIDREG) determine which primitives are highlighted.

2.4.4 Unhighlight Mode

This mode is used to unhighlight objects that have just been highlighted (see Example 2.5). UNHILITE mode draws using the color stored in VREG 0 (the current color value). Execute EXMODE UNHILITE.

2.5 Picking

Picking is a display controller function which allows you to specify a location on the screen, and then request the Model One to return the identity of the graphics primitive or primitives located at that place.

Before picking, you must specify the size of the pick aperture and the center of the pick aperture. When you execute a pick operation, the Model One identifies the graphics primitives which either overlap or fall within the pick aperture. Such identification is called a pick hit.

The Model One can identify either no pick hits, one pick hit, or multiple pick hits at one location. Each graphics primitive which is hit is labeled with a pick ID number and a segment ID number. These ID numbers are loaded into the registers and the pick buffer.

This is the picking process. The order of the first three steps can vary.

Step	Command
1) Specify the center of the pick aperture.	CLOAD 19 x,y
2) Specify the size of the pick aperture (or use default).	SETGL PICKAP
3) Specify the kind of information to store in the pick buffer (or use default).	SETGL PICKBUF
4) Enter pick mode.	EXMODE PICK
5) Reference the desired segment.	SEGREF n
6) Read the pick information.	RDPICK type
7) Enter normal draw or highlight mode.	EXMODE NORMAL or EXMODE HILITE

When you execute a pick operation, the Model One identifies the graphics primitives which either overlap or fall within a previously defined aperture. Such identification is called a pick hit. If no entities meet these conditions, there is no pick hit.

2.5.1 Specifying the Center of the Pick Aperture

Before picking, you must specify the pick aperture's center by loading its x,y values into Coordinate Register 19. Choose a center that is over or that crosses the graphics with information you want the Model One to read back and/or with the graphics you want to highlight.

Note that the aperture center can also be set with an interactive device by using the CMOVE command.

2.5.2 Specifying the Size of the Pick Aperture

The second step in the picking process is specifying the size of the pick aperture. Specify this square window with the SETGL PICKAP command. Specify the aperture in pixel units. The default aperture size on the Model One/80 is 20 x 20.

2.5.3 Specifying the Kind of Information to Store in the Pick Buffer

Third, specify the kind of information you want the Model One to store in the pick buffer (assuming a pick "hit" occurs) when you subsequently pick primitives. You can specify one of four items:

- Segment IDs
- Pick IDs (as set by the PICKID or INCPID command embedded within the segment definition)
- SEGID/PICKID pairs -- (default)
- Entire pick trees

  A pick tree is simply a list of pick IDs and segment IDs which reflects the hierarchy of each item picked. If an item is a child of some other segment, information can be derived about the parent segment as well. Pick trees are useful for applications using hierarchical structures and are described in Section 2.5.8.

To specify one of these options, use the SETGL PICKBUF command. In general, SEGIDs and PICKIDs require four bytes of storage. A tree requires eight bytes of SEGID/PICKID data for each level of segment nesting, plus a two-byte count of the nesting level.

After the pick operation, you can have the Model One read back the information you specified to store in the pick buffer with the RDPICK command (see Section 2.5.8).

2.5.4 Entering Pick Mode

Fourth, put the display controller into pick mode. To do this, use the EXMODE PICK command. At this point, the pick buffer is emptied and both SEGREG and PIDREG are set to -1.

2.5.5 Executing (Picking) Segments

The next step is executing, or picking, segments. Execute SEGREF and indicate the segment you want to pick. Note that nothing is drawn on the screen.

Every time the Model One sees a PICKID command, it knows it has encountered a pickable entity with a unique identification. When it detects that geometry commands are overlapping the aperture, it has made the first "hit" and it loads SEGREG and PIDREG with the current segment ID and the current pick ID.

Only the first pick recorded loads the registers. These registers tell the application what pick ID has been found and can later be used for highlighting, unhighlighting, and deleting.

Simultaneously, pick information is being recorded in a RAM area called the pick buffer. On a Model One/80 with a bulk memory card, this buffer area is 12K bytes.

In addition to being loaded into the appropriate registers, the first pick hit recorded is also stored in the pick buffer. The pick buffer contains this pick identity and the identities of all other pick hits recorded, unless the buffer overflows.

2.5.5.1 Pick Buffer Overflows

Any pick hits that occur after the buffer has filled are not recorded. Two counters are maintained to store two things -- the number of pick hits encountered and the number of pick hits recorded in the pick buffer. To determine if the pick buffer has overflowed, read these counters using the RDPICK PICKS command.

```
! SETGL IGNORE n
```

tells the display controller to ignore the first n pick hits. As a result, the application, after reading any pertinent information with RDPICK, can restart the pick operation and ignore all previously encountered pick hits if the pick buffer overflows. Use 0 to indicate "continue."

2.5.6 Reading Back Pick Information

Information stored in the pick buffer is read back with RDPICK. As discussed in Section 2.5.3, you have already specified the kind of information to store with the SETGL PICKBUF command. Depending on what you now request with parameters, RDPICK returns one of six types of information. Note that RDPICK parameters should correspond to SETGL PICKBUF parameters, although they need not always be exactly the same. For example:

SETGL PICKBUF parameter	stores	O.K. for
SETGL PICKBUF TREE	segment IDs, pick IDs, hierarchies	RDPICK with all options
SETGL PICKBUF SEGID	segment IDs	RDPICK PICKS, RDPICK SEGID
SETGL PICKBUF PICKID	pick IDs	RDPICK PICKS, RDPICK PICKID
SETGL PICKBUF SEGPID	segment IDs, pick IDs	RDPICK PICKS, RDPICK PICKID, RDPICK SEGID, RDPICK SEGPID

Note also that some readback with the RDPICK command is padded with minus ones.

On the following page is a list of all the RDPICK options along with a description of the information that the Model One returns with each. For each of the options, the startent parameter specifies the pick buffer entry to start reading from. If you specify 0 for startent, the Model One starts reading back from the next unread location. The totalent parameter specifies the total number of entries to read back. If you specify 0 for totalent, the Model One updates PIDREG and/or SEGREG, but does not return any data.

o RDPICK PICKS 0 0

Returns two 16-bit numbers, the number of pick hits encounted and the number of pick hits recorded in the buffer (the total number of pick hits does not include ignored hits). This data is gotten from two display list counters, not the pick buffer. Therefore, no registers are ever updated. With this option, information is always returned. If there is no hit, zeros are returned; if there is no hit recorded because the pick buffer is full, zeros are returned.

o RDPICK PICKID startent totalent

Returns the requested number of pick IDs. This option also loads the pick ID register (PIDREG) with the last pick ID read.

- If the number of pick IDs you request exceeds the number of pick IDs stored, the remaining output is padded with minus ones
- If pick IDs are not recorded, minus ones are returned.
- If no pick hit occurs, minus ones are returned.

o RDPICK SEGID startent totalent

Returns the requested number of segment IDs. This option also loads the segment ID register (SEGREG) with the last segment ID read.

- If the number of segment IDs you request exceeds the number of segment IDs stored, the remaining output is padded with minus ones.
- If segment IDs are not recorded, minus ones are returned.
- If no pick hit occurs, minus ones are returned.

o RDPICK SEGPID startent totalent

Returns the requested number of SEGID/PICKID pairs. This option also loads SEGREG with the last read segment ID and PIDREG with the last read pick ID.

- If the number of SEGID/PICKID pairs you request exceeds the number of pairs stored, the remaining output is padded with minus ones.
- If SEGID/PICKID pairs are not recorded, minus ones are returned.
- If no pick hit occurs, minus ones are returned.

o RDPICK TREEU startent totalent

Returns the requested number of pick trees in unpacked format.

Trees in unpacked format are returned as a one-word count indicating the tree depth (segment-nesting depth) at the time of the pick, followed by exactly 16 SEGID/PICKID pairs, even if some "pairs" are returned as two minus ones. This option also loads the SEGREG and PIDREG with the segment ID and pick ID of the bottom level of the last tree read.

- If the number of trees you request exceeds the number of trees stored, each remaining "tree" is returned as 00000 with 16 SEGID/PICKID pairs of minus ones.

- If hierarchies are not recorded, a tree of depth one is returned.

- If no pick hit occurs, minus ones are returned.

o RDPICK TREEP startent totalent

Returns the requested number of trees in packed format.

In packed format, trees are returned as a one-word count indicating the tree depth (segment-nesting depth) at the time of the pick, followed by x SEGID/PICKID pairs (where x is the nesting depth). This option also loads SEGREG and PIDREG with the segment ID and pick ID of the bottom level of the last tree read.

- If the number of trees you request exceeds the number of trees stored, the remaining "trees" are returned as 00000 (no SEGID/PICKID pairs).

- If hierarchies are not recorded, a tree of depth one is returned.

- If no pick hit occurs, minus ones are returned.

2.5.6.1 Using the RDPICK Command

The following example illustrates a practical use of specifying 0 for the startent parameter.

```
! RDPICK SEGPID 2 1        ;Read back the second entry in the pick
                            buffer.
! RDPICK SEGPID 0 1        ;Read back the third entry in the pick
                            buffer (the next unread location).
! RDPICK SEGPID 0 2        ;Read back the fourth and fifth entries
                            in the pick buffers.
```

2.5.6.2 Using the RDPICK Command Only for Updating Registers

You can use RDPICK to update SEGREG and PIDREG without also getting readback. To use RDPICK for updating registers, execute the command with a type parameter other than PICKS. Request that no data be returned by specifying "0" for the last parameter, totalent. For example:

- RDPICK PICKID 1 0 updates the pick ID register with the first entry.
- RDPICK SEGID 2 0 updates the segment ID register with the second entry.
- RDPICK SEGPID 12 0 updates both registers with the 12th entry.

2.5.7 Entering Normal Draw or Highlight Mode

At this point in the picking process you should return to normal draw mode or highlight mode so that graphics can again be displayed or highlighted. Return to normal draw mode with the EXMODE NORMAL command or return to highlight mode with the EXMODE HILITE command.

2.5.8 Using Picking

An application that makes use of hierarchical local display lists generally needs to record entire pick trees so that the host can know where a pick occurred in a nested segment. Hence, a typical pick operation looks like the one shown on the following page.

Example 2.4 Picking

```
! CLOAD 19 -420,-160        ;Load CREG 19 to specify the center of the
                             pick aperture.
! SETGL PICKAP 10 10        ;Set the pick aperture size to 20 x 20
! SETGL PICKBUF TREE        ;Specify to save entire trees in pick
                             buffer.
! EXMODE PICK               ;Enter pick mode.
! SEGREF 1                  ;Execute (pick) Segment 1, the top level
                             segment.
! RDPICK TREEP 1 2          ;Read back tree hierarchy in packed format,
                             starting with the first entry and reading
                             back information on two entries. Update
                             registers.

  00001
 0000000001 0000000020
  00002
 0000000001 0000000020
 0000000002 0000000030
! EXMODE NORMAL             ;Restore execution mode to normal draw.
```

In the above example, the graphics commands to be searched are stored in Segment 1, the segment tree specified in the SEGREF command. Note that the SEGREF command always executes to completion; the full segment is traversed regardless of both the number of pick hits and whether or not the pick buffer is full. The first pick encountered loads the SEGREG and PIDREG to facilitate local highlighting.

2.6 Local Highlighting

Highlighting is the process of drawing graphics primitives from a given segment with a given pick ID in a specified highlight color. The highlight color can be used to make these entities stand out or even blink with use of the blink table.

The segment ID and pick ID of the primitives to highlight are contained in SEGREG and PIDREG. These registers are updated automatically during picking when there is a pick hit; or, you can update them interactively with either the RDPICK command and the SETGL command, both with the appropriate options. The highlight color is taken from VREG 45 which you must load before highlighting.

Unhighlighting is the inverse of highlighting. Unhighlighting is redrawing specified primitives based on the current color, taken from VREG 0.

Enter highlight and unhighlight modes with the EXMODE HILITE and EXMODE UNHILITE commands. Once you select either highlight or unhighlight mode, only primitives of the specified pick ID contained in the specified segment are drawn. All other commands are evaluated but they do not cause writes to image memory.

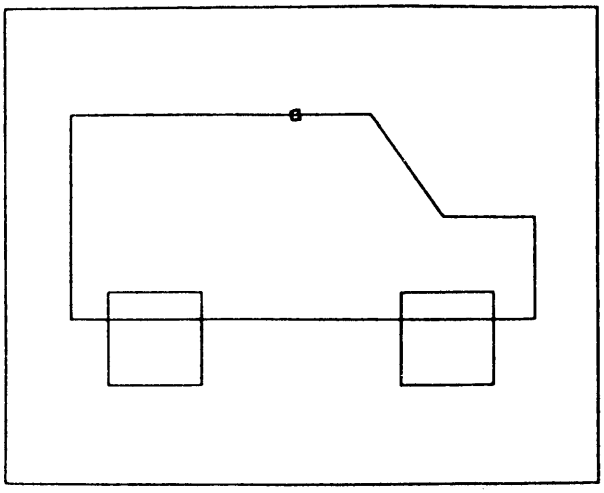
A local display list application generally highlights data that has just been picked. For a typical highlighting command stream, see Example 2.5.

The actual highlight operation begins when the segment ID and pick ID match those contained in the SEGREG and PIDREG and it continues until a PICKID command alters the PIDREG or until the segment ends. If there are nested segments within a highlighted segment, the nested segments are also highlighted if their visibility attribute is ON.

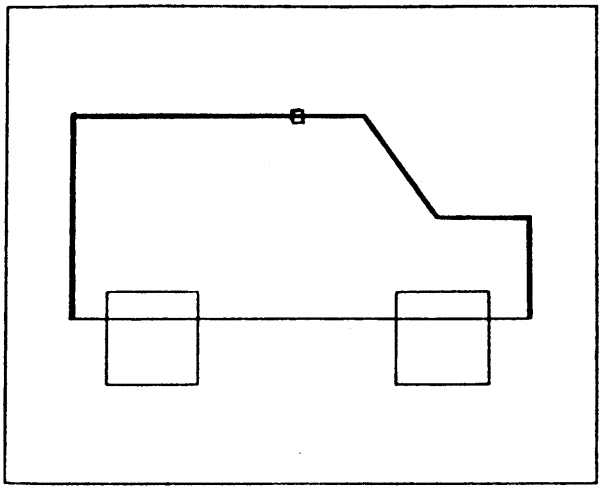
Example 2.5 Highlighting

```
! EXMODE NORMAL          ;Set execution mode to normal draw.
! SEGREF 1               ;Execute (draw) Segment 1 (See the figure
                          labeled NORMAL DRAW).
! CLOAD 19 0,280         ;Load CREG 19 to specify the center of the
                          pick aperture.
! SETGL PICKBUF SEGPID   ;Specify to store segment ID/pick ID pairs
                          (default).
! VLOAD 45 48,0,0        ;Load VREG 45 with highlight color, red.
! EXMODE PICK            ;Enter pick mode.
! SEGREF 1               ;Execute (pick) Segment 1.
! EXMODE HILITE          ;Enter highlight mode.
! SEGREF 1               ;Execute (draw) Segment 1, highlighting the
                          primitives of the first pick hit (See
                          figure labeled HILITE)
! EXMODE UNHILITE        ;Set execution mode to unhighlight.
! SEGREF 1               ;Execute Segment 1: draw in normal color the
                          geometry that has ben picked.
! EXMODE NORMAL          ;Resume normal draw mode.
```

NORMAL DRAW



HILITE



2.7 Transformations

The primitives within segments are defined in a 65,536 by 65,536 world coordinate system (WCS) ranging from (-32768,-32768) to (32767,32767). The device coordinate system (DCS) is the coordinate system of the display screen. DCS coordinates are calculated from WCS coordinates with a transformation applied.

Applications usually use a larger WCS area than can normally be seen at one time without applying transformations; they also often need to translate and rotate geometry. Raster's Display List Firmware allows scaling, translating, and rotating with a command that modifies, not the graphics primitives, but the display of those primitives.

How graphics are displayed is determined by a current transformation matrix which specifies the portion of the WCS which is visible and any scaling or rotation applied. The system keeps a default transformation matrix (3 x 2); you can modify this matrix with the XFORM2D command.

With one execution of XFORM2D, you can modify the current transformation matrix and achieve any combination of translation, rotation, and scaling. You define a transformation with XFORM2D by first specifying a transformation type.

2.7.1 Types of Transformations

You can define the following four types of transformations with the XFORM2D command.

XFORM2D REL	Specifies a 2 x 2 matrix which performs relative scaling and/or rotation centered about the current point; matrix elements are concatenated with the previous transformation.
XFORM2D ABS	Specifies a 3 x 2 matrix which performs absolute scaling with translation, rotation centered about the current point with translation, or both with translation.
XFORM2D XLATE	Specifies translation of the WCS origin to the current point.
XFORM2D RESET	Specifies a return to the default transformation (the identity).

2.7.1.1 What You Have to Specify

The parameters which you specify for the XFORM2D command vary according to the type of transformation.

The RESET and XLATE types require only one parameter, type.

The REL and ABS types require a reserved parameter, which must equal zero.

2.7.1.2 Default Transformation: The Identity

The default transformation is the identity, which specifies no scaling, no rotation, and no translation.

```
     x basis vector = (1, 0)
     y basis vector = (0, 1)
displacement vector = (0, 0)
```

To return to the default identity, use the RESET type of the XFORM2D command.

2.7.1.3 Translation With No Scaling or Rotation: XFORM2D XLATE

Use the XLATE type of the XFORM2D command to specify a translation with no scaling or rotation. Before executing the command, move the current point with the MOVABS or MOVREL command to reflect the translation you wish to effect (i.e., the displacement vector).

2.7.1.4 Absolute Transformation: XFORM2D ABS

For the absolute transformation type, ABS, you must specify a 3 x 2 matrix, consisting of the transformed x basis vector and y basis vectors, as well as the displacement vector.

Listed below are the sets of vectors (or 3 x 2 matrices) which describe scaling with translation, rotation with translation, and scaling followed by rotation with translation.

To specify no translation, use zeros in the last row.

To concatenate different transformations, apply the rules of matrix multiplication.

Scale by scale factors Sx and Sy
Translate by displacement vector D

```
     x basis vector =  (Sx, 0)
     y basis vector =  (0, Sy)
displacement vector =  (Dx, Dy)
```

Rotate by A degrees
Translate by displacement vector D

```
     x basis vector =  (cos(A), sin(A))
     y basis vector =  (-sin(A), cos(A))
displacement vector =  (Dx, Dy)
```

Scale by scale factors Sx and Sy
Rotate by A degrees
Translate by displacement vector D

```
     x basis vector =  (Sx  cos(A),  Sy  sin(A))
     y basis vector =  (-Sx  sin(A), Sy  cos(A))
displacement vector =  (Dx, Dy)
```

2.7.1.5 Relative Transformation: XFORM2D REL

For the relative transformation type, REL, you must specify a 2 x 2 matrix. This matrix includes the x and y basis vectors as listed above (2.7.1.4), but does not include the displacemant vector. To better illustrate the required input for the XFORM2D command, Tables 2.1 and 2.2 are provided. Table 2.1 gives the exact input for eight relative rotations.

Table 2.1 XFORM2D: Eight Relative Rotations

Rotate by	Command			COS	SIN	-SIN	COS
30 degrees	XFORM2D	REL	0	.866	.5	-.5	.866
45 degrees	XFORM2D	REL	0	.707	.707	-.707	.707
60 degrees	XFORM2D	REL	0	.5	.866	-.866	.5
90 degrees	XFORM2D	REL	0	.0	1	-1	.0
120 degrees	XFORM2D	REL	0	-.5	.866	-.866	-.5
135 degrees	XFORM2D	REL	0	-.707	.707	-.707	-.707
150 degrees	XFORM2D	REL	0	-.866	.5	-.5	-.866
180 degrees	XFORM2D	REL	0	-.1	.0	-.0	-.1

Table 2.2 shows the exact input for three absolute scalings.

Table 2.2 XFORM2D: Three Absolute Scalings

Scale by	Command		Scale Factor Times x	0	0	Scale Factor Times y	Trans-lation Factor Times x	Trans-lation Factor Times y
2	XFORM2D	ABS	0	2	0	0	2	0	0
4 1/2	XFORM2D	ABS	0	4.5	0	0	4.5	0	0
5	XFORM2D	ABS	0	5	0	0	5	0	0

2.7.2 Reading Back and Using the Current Transformation

The read back command RDXFORM returns the current transformation matrix in the same format as that used with the XFORM2D ABSOLUTE command.

You can use the current transformation matrix to perform calculations locally within the Model One. Do this with the XMOVE command which maps a specified x,y coordinate from one coordinate space into another coordinate space. With the appropriate option, the command maps either from WCS to DCS or from DCS to WCS.

2.7.3 How Transformations Affect the Current Point

When you execute the XFORM2D command, the current point is set to 0,0. This ensures that the current point is always at the origin of the new transformation.

Example 2.6 Transformations and the Current Point

```
! MOVABS 20,30          ;Move the current point to 20,30.
! POINT                 ;Draw a point at 20,30.
! XFORM2D ABS 0 2 0 0 20 0
                        ;Define an absolute transformation which
                         scales by a factor of 2.
! READCR 0              ;Read the contents of CREG 0 (current point).

00000  00000            ;The Model One returns 0,0. The current point
                         has been reset by the transformation.
! POINT                 ;Draw a point at 0,0.
```

2.8 Stacking

Two display list commands, PUSH and POP, allow applications programs to store certain items in internal RAM and then to restore these items.

With the PUSH command, you can save registers, flags, variables, and transformation matrices. These items are placed on a stack until you want to restore them by executing the POP command.

You must pop stacked items in the exact opposite order as they were pushed. The PUSH and POP commands do not perform any type checking. For example, the following sequence may produce unpredictable results:

```
! PUSH XFORM 0
! POP CREG 0
```

The Model One/80 has 2,000 stack locations. The PUSH and POP commands perform stack overflow and underflow checking. An error is generated if you exceed the stack bounds. Pushing a 2-D transformation matrix uses 11 locations, pushing a CREG uses two locations, pushing a VREG uses three locations, and pushing a variable uses one location.

2.8.1 Pushing and Popping Transformations

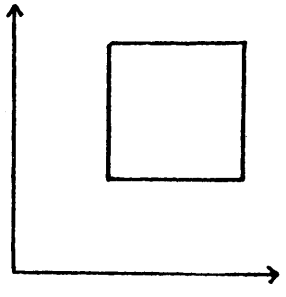
One of the options for the PUSH and POP commands is to save and restore the current 2-D transformation. Another option is to save and restore the current 2-D transformation along with the current point. The syntax for these commands is shown on the following page.

Syntax	Description
PUSH XFORM XF2D	Push the current 2-D transformation onto the stack.
PUSH XFORM XF2DCP	Push the current 2-D transformation and the current point onto the stack.
POP XFORM XF2D	Pop the current 2-D transformation from the stack.
POP XFORM XF2DCP	Pop the current 2-D transformation and the current point from the stack.

Example 2.7 illustrates the use of PUSH/POP XFORM XF2D. Note that PUSHXFORM XF2D command does not save the current point.

Example 2.7 Pushing and Popping a Transformation

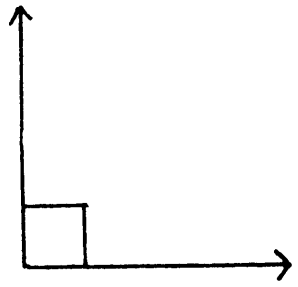
```
! SEGDEF 10              ;Begin definition of Segment 10, which draws
                          a square.
! DRWREL 0 20
! DRWREL 20 0
! DRWREL 0 -20
! DRWREL -20 0
! SEGEND                 ;End definition of Segment 10.
! MOVABS 30,30           ;Move current point to 30,30.
! PUSH XFORM XF2D        ;PUSH the current 2-D transformation onto the
                          stack.
! XFORM2D REL 0 2 0 0 2
                         ;Define a relative transformation which
                          scales by a factor of 2.
! SEGREF 10              ;Execute (draw) Segment 10.
```



Continued

Example 2.7, continued

```
! POP XFORM XF2D          ;Pop 2-D transformation from stack.
! SEGREF 10               ;Execute Segment 10.
```



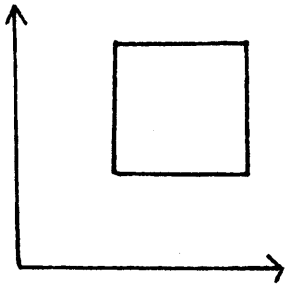
```
! READCR 0                ;Read CREG 0 (current point).

00000  00000              ;The current point has been set to 0,0.
! POP XFORM XF2D          ;Pop 2-D transformation from stack.
```

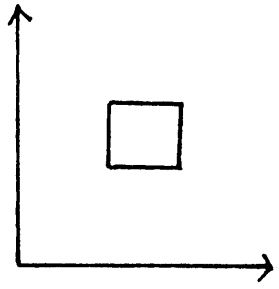
Example 2.8 is similar to the previous example, except that this time the current point is saved along with the transformation (PUSH/POP XFORM XF2DCP).

Example 2.8 Pushing and Popping a Transformation and the Current Point

```
! XFORM2D RESET          ;Reset 2-D transformation to the identity.
! MOVABS 30,30           ;Move the current point to 30,30.
! PUSH XFORM XF2DCP      ;Push the current 2-D transformation and
                          the current point onto the stack.
! XFORM2D REL 0 2 0 0 2
                         ;Define a realtive transformation which
                          scales by a factor of 2.
! SEGREF 10              ;Execute (draw) Segment 10 (defined in
                          preceding example).
```



```
! POP XFORM XF2DCP       ;Pop the 2-D transformation and the
                          current point from the stack.
! SEGREF 10              ;Execute (draw) Segment 10.
```



Continued

Example 2.8, continued

```
! READCR 0                ;Read CREG 0 (current point).

00030  00030              ;The current point has been saved and
                           restored.
```

2.9 Display List Firmware Commands By Functional Category

The Display List commands can be divided into the following functional categories:

- o Segment creation and modification commands
- o Attribute setting commands
- o Segment execution commands
- o Readback commands

Other related groups of commands include:

- o Transformation commands
- o Stacking commands
- o Picking commands

Segment creation and modification commands create and edit segment structures. Attribute setting commands are used to set and modify segments attributes (visibility and pickability) and to assign commands to pick IDs.

The segment execution commands set the display controller execution mode and execute segments. Readback commands return information about the list structure. Transformation commands perform translations, scaling, and rotations. Finally, the stacking commands save and restore registers, flags, variables, and transformations.

A complete list of Model One/80 Display List commands, functional category, is given in Table 2.3. All these commands are documented fully in the Model One/80 Command Reference (Revision 1.0).

Table 2.3 Display List Firmware Commands

Segment-creation and -modification Commands

SEGINI (Initialize All Segments)
SEGDEF (Begin A Segment Definition)
SEGAPP (Segment Append)
SEGEND (Segment End)
SEGDEL (Segment Delete)
SEGCOP (Segment Copy)
SEGREN (Segment Rename)
DELPID (Delete Pick Identification Number)

Attribute-setting Commands

INCPID (Assign Pick Identification Number)
PICKID (Assign Pick Identification Number)
SETATR (Set Attribute)
SETGL (Set Global Parameter)

Segment-execution and Picking Commands

EXMODE (Set Execution Mode of Display Controller)
SEGREF (Execute a Segment)

Readback Commands

RDPICK (Read Back Pick Information)
RDPID (Read Back List of All Commands with specified segment ID and pick ID)
RDREG (Read Back Segment ID and Pick ID Registers)
RDXFORM (Read Back Current Transformation)
SEGINQ (Segment-attribute Inquiry)
SYSTAT (Read Back Defined-segment or Existing-RAM Information)

Transformation Commands

XFORM2D (Transform All 2-D Coordinates)
XMOVE (Map One (x,y) Location)

Stacking Commands

PUSH (Save Specified Current State)
POP (Restore "PUSH" State)

Appendix A

Converting from Model One/25, Model One/40, Model One/60 Display List Firmware

to

Model One/10, Model One/80 Display List Firmware

A. Converting from Model One/25, Model One/40, Model One/60 Display List Firmware to Model One/10, Model One/80 Display List Firmware

The new Model One/10 and Model One/80 Display List Firmware Product is an enhancement of the previous version of The Raster Technologies Display List Firmware for the Model One/25, Model One/40, and Model One/60. The two products, while not compatible, are very similar and older software can be easily converted with the routines described in this section.

First, the few differences are discussed. These differences are:

- Pick IDs and Segment IDs are 32-bit integers with the Model One/10, Model One/80 Display List Firmware. With the Display List Firmware on the Model One/25, Model One/40, and Model One/60, however, these entities are 16-bit integers.

- With the new Display List Firmware, you can nest transformations. Such nesting is not available with previous Display List commands.

- The Model One/10 and Model One/80 Display List Firmware does not use "viewports." Instead, windows into WCS (or MCS) are specified by the WINDOW, XFORM2D, WRMASK, and SEGREF commands. Earlier Display List Firmware uses the DEFVW commands to define viewports.

- The new XFORM2D command defines transformations that can include any combination of translation, scaling, and rotation. These operations are performed with the DEFVW command on the Model One/25, Model One/40, and Model One/60.

- The XMOVE command is also new for the Model One/10, Model One/80 Display List Firmware Product. This command maps one specified x,y coordinate, usually the current point, from either WCS to DCS or from DCS to WCS.

- Another related command, new for the Model One/10 and Model One/80 Display List Firmware, is the read back command RDXFORM. It returns the current specified transformation, as you defined it.

- With the enhanced firmware, drawing, picking, and highlighting of "viewports" are not performed by executing the combination of REDRAW, PICKCR and HILITE commands, as is done with the older firmware. Instead, the display controller is placed in one of four modes (NORMAL for drawing, PICK for picking, HILITE/UNHILITE for highlighting and unhighlighting) and the top level segment of a tree is referenced using the SEGREF command.

- With the new Model One/10 and Model One/80 firmware, commands involving coordinate registers are not resolved at segment definition time; that is, the system does not look at the contents of registers during segment definition.

- Even though it is not recommended, with the new Display List Firmware you can use absolute MOVEs and DRAWs within nested segments. This is not possible with the other Display List Firmware.

The remaining part of this appendix is a command-by-command guide to the Model One/25, Model One/40, Model One 60 Display List commands and the enhanced Model One/10, Model One/80 Display List commands. This section also includes directions for adaptation.

DEFVW

This command is not in the new product. Using the new product, you express viewports in terms of a clipping window, a background color value, a write mask, a transformation, and a top level segment. An application can create a "viewport" by defining a segment with all of the necessary view information.

DELPID

This command is the same in both the Model One/10, Model One/80 Display List Firmware Product and the Model One/25, Model One/40, Model One/60 Display List Firmware Product.

HILITE

This command in the old Display List Firmware is replaced by EXMODE HILITE/UNHILITE in the new product. The highlight color must be stored in Value Register 45 of the Model One/10 and Model One/80. To emulate the HILITE command, use a sequence such as the following:

```
HILITE 80 1 40          ;Redraw View 80, highlighting it with
                         vreg 40
```

is replaced with:

```
!VMOVE 45,40            ;Load the highlight color.
!EXMODE HILITE          ;Enter hilite mode.
!SEGREF 80              ;Execute (draw in highlight color) the
                         "viewport" segment (see DEFVW).
!EXMODE UNHILITE        ;Enter unhilite mode.
!SEGREF 80              ;Execute (unhilight) the segment.
!EXMODE NORMAL          ;Enter normal draw mode.
```

In general:

HILITE view 1 vreg	HILITE view 0 vreg
is replaced by:	is replaced by:
VMOVE 45,vreg EXMODE HILITE SEGREF view	VMOVE 45,vreg EXMODE UNHILITE SEGREF view

PICKCR

This command in the older firmware is replaced by EXMODE PICK in the new Display List Firmware Product. The center of the pick aperture must be contained in Coordinate Register 19 in the Model One/10 and Model One/80. The original searchflag function can be emulated using the new RDPICK command.

The following sequence emulates PICKCR 80 xx 0:

```
!CMOVE 19 xx          ;Set the center of the pick aperture.
!EXMODE PICK          ;Enter pick mode.
!SEGREF 80            ;Execute (pick) the "viewport"
                       segment (see DEFVW); update SEGREG and PIDREG.
                       SEGREG/PIDREG now contain information from
                       the first pick.
!EXMODE NORMAL        ;Enter normal draw mode.
```

The following sequence emulates PICKCR 80 xx 1:

```
!CMOVE 19 xx          ;Set the center of the pick aperture.
!EXMODE PICK          ;Enter pick mode.
!SEGREF 80            ;Execute (pick) the "viewport" segment
                       (see DEFVW); update SEGREG and PIDREG.
                       SEGREG/PIDREG now contain information
                       from the first pick.
!EXMODE NORMAL        ;Enter normal draw mode.
!INCPID               ;Load SEGREG/PIDREG with information
                       from the second pick.
!INCPID               ;Load SEGREG/PIDREG with information
                       from the third pick.
```

RDPID

The format of the readback is slightly different. This command is byte-oriented in the new product, instead of word-oriented, as it is in the earlier product. With the Model One/10 and Model One/80 Display List Firmware, (n,nray) is returned, where n is a count of bytes (instead of words) and nray is an array of n bytes (instead of words).

With the older Display List Firmware, 8-bit opcodes and parameters are returned as 16-bit quantities (the high byte is padded with a zero). In contrast, the Model One/10 and Model One/80 Display List Firmware returns segment contents in the exact form that it was defined and stored (8-bit opcodes and parameters are read back as 8-bit quantities).

RDREG

This command is the same in both the Model One/10, Model One/80 Display List Firmware Product and the Model One/25, Model One/40, Model One/60 Display List Firmware Product.

RDTREE

This command is replaced in the new Display List Firmware by a variant of the RDPICK command. To emulate the RDTREE command, an application should:

- Specify that trees are to be recorded in the pick buffer during picking. Use SETGL PICKBUF TREE.
- Perform a pick operation. Use EXMODE PICK and SEGREF.
- Execute RDPICK TREEU 0 1 to read back the next tree in the pick buffer in unpacked format.

When you read back unpacked trees, a 16 x 2 array (instead of 9x2, the older format) of segment IDs and pick IDs is returned.

REDRAW

This command is not part of the new Display List Firmware. Segments are drawn by executing them with the execution mode set to NORMAL. To emulate the REDRAW command with View 80 as before, for example, execute:

```
!EXMODE NORMAL          ;Enter normal draw mode.
!SEGREF 80              ;Execute (draw) the "viewport" segment
                         (see DEFVW).
```

If you want to clear the window to the background color before drawing that window, execute an explicit CLEAR instruction after you specify the clipping window.

SEGAPP

This command is unchanged except that the segment number is a 32-bit integer in the new Display List Firmware Procuct.

SEGCOP

This command is unchanged except that the segment numbers are now 32-bit integers in the new Display List Firmware Procuct.

SEGDEF

This command is unchanged except that the segment numbers are now 32-bit integers in the new Display List Firmware Procuct.

SEGDEL

This command is unchanged except that the segment numbers are now 32-bit integers in the new Display List Firmware Procuct.

SEGEND

This command is the same in both the Model One/10, Model One/80 Display List Firmware Product and the Model One/25, Model One/40, Model One/60 Display List Firmware Product.