Halftones and Screens

Adobe® Developers Association

9 October 1997

Technical Note #5602

LanguageLevel 3

Adobe Systems Incorporated

Corporate Headquarters
345 Park Avenue
San Jose, CA 95110-2704
(408) 536-6000

Eastern Regional Office
24 New England
Executive Park
Burlington, MA 01803
(617) 273-2120

Adobe Systems Europe Limited
Adobe House, Mid New Cultins
Edinburgh EH11 4DU
Scotland, United Kingdom
+44-131-453-2211

Adobe Systems Japan
Yebisu Garden Place Tower
4-20-3 Ebisu, Shibuya-ku
Tokyo 150 Japan
+81-3-5423-8100

PN LPS5602

Adobe Systems Incorporated

Contents

Adobe Systems Incorporated

Figures

Adobe Systems Incorporated

Tables

Adobe Systems Incorporated

Examples

Adobe Systems Incorporated

Preface

This Document

This is the original release for *Halftones and Screens,* a document that provides a detailed description of the LanguageLevel 3 extensions to halftones, halftone screens, and halftone cells. These extensions allow developers to take advantage of a new screen mechanism that allows the printing of up to 256 levels of gray on mid-range devices, and a new halftone type that allows the printing of over 256 levels of gray on high-end devices. Both of these new features allow for more photorealistic gray tones in print and display media.

Intended Audience

This document is written for software developers who are interested in learning about halftones, halftone screens, and halftone cells, or adding these capabilities to an application that supports PostScript® display or printing devices.

It is assumed that the developer is already familiar with how halftones, halftone screens, and halftone cells work in previous levels of the PostScript language.

Organization of This Document

Section 1, “Halftones and Screens,” gives the history and use of halftones and halftone operators available in LanguageLevel 1. The addition of halftone dictionaries, threshold arrays, accurate screens, and supercells in LanguageLevel 2 is described. Finally, the extensions to the language for LanguageLevel 3 are introduced.

Section 2, “Supercells,” gives an overview of halftone screens and halftone cells. The changes to supercells for halftone screens, including the use of the new **MaxSuperScreen** user parameter, is described. Example usage of this new user parameter is given. The benefits of using supercells is also presented.

Section 3, “Halftone Dictionaries,” covers changes to older halftone dictionaries and gives an overview of the new halftone dictionaries. A detailed description of the new **HalftoneType 16** halftone dictionary is presented. An example use of the Type 16 halftone is shown. Benefits of using type 16 halftones are also presented.

Section 4, “Tips and Techniques,” gives helpful information on using halftones, halftone screens, and halftone cells, and selecting the best types for specific application needs.

Related Publications

Supplement: PostScript Language Reference Manual (LanguageLevel 3 Specification and Adobe PostScript 3™ Version 3010 Product Supplement), available from the Adobe® Developers Association, describes the formal extensions to the PostScript language that have occurred since the publication of the PostScript Language Reference Manual, Second Edition. This supplement also includes all LanguageLevel 3 extensions available in version 3010.

PostScript Language Reference Manual, Second Edition (Reading, MA: Addison-Wesley, 1991) is the developer’s reference manual for the PostScript language. It describes the syntax and semantics of the language, the imaging model, and the effects of the graphical operators. See “Halftones,” Section 6.4 for more information on halftone concepts and terms.

Statement of Liability

THIS PUBLICATION AND THE INFORMATION HEREIN IS FURNISHED AS IS, IS SUBJECT TO CHANGE WITHOUT NOTICE, AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY ADOBE SYSTEMS INCORPORATED. ADOBE SYSTEMS INCORPORATED ASSUMES NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR INACCURACIES, MAKES NO WARRANTIES OF ANY KIND (EXPRESS, IMPLIED, OR STATUTORY) WITH RESPECT TO THIS PUBLICATION, AND EXPRESSLY DISCLAIMS ANY AND ALL WARRANTIES OF MERCHANTABILITY, FITNESS FOR PARTICULAR PURPOSES, AND NONINFRINGMENT OF THIRD-PARTY RIGHTS.

Halftones and Screens

1 Halftones and Screens

1.1 Overview of Halftones and Screens

Prior to the emergence of digital laser printers and imagesetters, halftoning was a laborious manual process involving pre-printed halftone screens. With the advent of page description languages such as the Adobe® PostScript® language, the concern of the pre-press industry turned to digital halftoning techniques, where the halftone screens are generated directly by the laser printer or imagesetter.

For applications such as color separation, the crucial requirement of the pre-press business is correctly matched screen sets. A screen set for the traditional four-color process model must contain four plates, one for each of the colors. A matched screen set means that the frequency (lines per inch or lines per millimeter) and the angles of the halftone screens must be designed such that no moiré patterns are produced when all four screens are superimposed on top of each other.

Table 1 provides a brief description of the various kinds of halftoning supported in all levels of the PostScript language. LanguageLevel 2 introduced the notion of using a dictionary containing all the halftone parameters as the argument to the new **sethalftone** operator. LanguageLevel 1 **setscreen** and **setcolorscreen** operators have basically been replaced by the use of **sethalftone** with its dictionary argument as the recommended method to specify halftone screens.

The table is broken down into three basic sections, one for each level of the PostScript language. For each level, the associated operators and halftone types, if supported, are given, as well as a brief description of each operator/dictionary combination.

Table 1 *Halftones in the PostScript Language*

Language Level	Operators	Halftone Type	Description
1	**setscreen, currentscreen**	n/a	Specifies frequency, angle, and spot function
1	**setcolorscreen, currentcolor-screen**	n/a	Same as for setscreen, but with four sets of parameters
2	**sethalftone, currenthalftone**	1	Same as for setscreen
2	**sethalftone, currenthalftone**	2	Same as for setcolorscreen
2	**sethalftone, currenthalftone**	3	An 8-bit threshold array in a string
2	**sethalftone, currenthalftone**	4	Four 8-bit threshold arrays
2	**sethalftone, currenthalftone**	5	An arbitrary number of Type 1, Type 3, Type 6, Type 9, Type 10, Type 16, or Type 100 halftones
3	**sethalftone, currenthalftone**	6	Similar to Type 3 except threshold array comes from a file
2, 3	**sethalftone, currenthalftone**	9	Contains proprietary data; printer dependent
2, 3	**sethalftone, currenthalftone**	10	An 8-bit threshold array with a non-zero angle
3	**sethalftone, currenthalftone**	16	Similar to Type 10, but with 16-bit array values
2, 3	**sethalftone, currenthalftone**	100	Similar to Type 9, with implementation-defined keys; printer dependent

Some definitions will help in the understanding of Table 1 and in the topics covered in this document:

- Halftone Screen and Halftone Cell: a halftone screen is a uniform rectangular grid of halftone cells. Each device pixel belongs to one halftone cell of this grid; a halftone cell usually contains many device pixels. Certain device pixels combine to make a halftone cell of the grid. All of the halftone cells of the grid together make the halftone screen.

- Frequency: this parameter to **setscreen** and **setcolorscreen** specifies the number of halftones cells per inch within a halftone screen grid. This parameter is also a key for Type 1 and 2 halftone dictionaries.

- Angle: this parameter specifies the orientation of the grid lines (of the halftone screen) relative to the device coordinate system. This parameter is a key for Type 1 and 2 halftone dictionaries.

- Spot Function: this parameter specifies the order in which pixels change from black to white for increasing gray levels (or from off to on for increasing intensity of color). The resulting halftone cell is always square. This parameter is also a key for Type 1 and 2 halftone dictionaries.

- Threshold Array: this array directly controls individual device pixels in a halftone cell. In simpler terms, a threshold array contains data that indicates whether or not to paint a specific pixel at the current color value. Like a sampled image, it is a rectangular array of pixel values defined entirely in device space. The threshold values always occupy 8 bits each, except in the case of Type 16 halftones.

1.2 Halftones in LanguageLevel 1

LanguageLevel 1 halftones were implemented using the **setscreen** operator (and the **setcolorscreen** operator for those Level 1 devices that supported color).

Because of the manner in which the halftone cells were defined, it was not always possible to obtain the correct angle for a particular separation, even though the angle was specified explicitly. This restriction arose from the use of *rational tangents* – the corners of a halftone cell had to align precisely with device pixels. This restriction limited the available angles that a cell could adopt.

One additional restriction was that only one halftone screen could be defined, regardless of the number of color planes. This meant that each separation would have to use the same screen. Some early color printers, though, supported the **setcolorscreen** operator for allowing a unique screen for each color plane.

A once popular use for spot functions was to create patterns, such as basket weaves, bricks, and so on, that tile a region. The spot function in such an application of halftone cells actually created the desired pattern. Unfortunately, there were types of devices in which this method did not work at all. LanguageLevel 2 halftones, discussed in the next section, introduced the **Pattern** color space, that performs the same function in a device-independent fashion. **Pattern** color spaces are now the recommended way to tile regions with patterns.

Note *For more information on the use of the **setscreen**, **currentscreen**, **setcolorscreen**, and **currentcolorscreen** operators, halftones, halftone screens, and halftone cells, see Section 6.4 of the PostScript Language Reference Manual, Second Edition.*

1.3 Halftones in LanguageLevel 2

LanguageLevel 2 introduced new halftoning capabilities in response to customer requirements. While supporting the LanguageLevel 1 **setscreen** and **setcolorscreen** operators for backward compatibility, LanguageLevel 2 introduced more flexible halftoning schemes using halftone dictionaries and the **sethalftone** operator. LanguageLevel 2 introduced five different kinds of halftone dictionaries.

The following is a list of halftone dictionaries that are available in LanguageLevel 2:

- Type 1: defined by a single halftone screen which uses the keys **Frequency**, **Angle**, and **SpotFunction**. These are the same parameters used by the **setscreen** operator.
- Type 2: defined by 4 separate halftone screens, each of which is defined as above for a Type 1 halftone. These are the parameters that would normally be used by the **setcolorscreen** operator.
- Type 3: defined by a single halftone screen that uses a single, eight-bit threshold array to hold the halftone cell information.
- Type 4: defined by four separate halftone screens that each use an eight-bit threshold array. The four screens make it possible to have one screen for each of the four typical color plates in a color separation.
- Type 5: defined by an arbitrary number of halftone screens, one for each color component, including primary and spot colors. Each of these screens is a Type 1, Type 3, Type 6, Type 9, Type 10, Type 16, or Type 100 dictionary. Type 5 halftones are intended for separations of spot colors and process colors in the same print job.

Note *For more information on the halftone types **1** through **5**, see Section 6.4 of the PostScript Language Reference Manual, Second Edition.*

Example 1 illustrates the use of a type 3 halftone – a threshold array whose data comes from a string within the halftone dictionary. The code shown here sets up a halftone type 3 screen and then uses it to set the gray levels for five vertical stripes that are drawn in five different shades of gray.

Example 1 *Type 3 Halftone with Threshold Arrays in Strings*

```
%!PS
/inch {72 mul} def
<<  %start halftone dictionary
    /HalftoneType 3 % 8-bit threshold data from str.
    /Width 20 % 20 pixel wide cell
    /Height 40 % 40 pixel high cell
    /Thresholds   <
        58654B0B4EF98316ED092395D633B7297C641223
         ... % threshold data goes here
         59D567EA1EACD159B8804EB9E237A18BF4106535
    >
>> sethalftone
% set origin for first stripe, set the gray value,
% draw the stripe
1 inch 1 inch translate
0.25 setgray
0 0 1.3 inch 9 inch rectfill
% set origin for second stripe, set the gray value,
% then draw the stripe
1.3 inch 0 translate
0.3333 setgray
0 0 1.5 inch 9 inch rectfill
% set origin for the third stripe, set the gray value,
% then draw the stripe
1.3 inch 0 translate
0.50 setgray
0 0 1.3 inch 9 inch rectfill
% set origin for the fourth stripe, set the gray value,
% then draw the stripe
1.3 inch 0 translate
0.6667 setgray
0 0 1.3 inch 9 inch rectfill
% set origin for the last stripe, set the gray value,
% then draw the last stripe
1.3 inch 0 translate
0.75 setgray
0 0 1.3 inch 9 inch rectfill
showpage
```

In addition to new halftone dictionaries and threshold arrays, LanguageLevel 2 created a new implementation of halftone screens based on supercells. In LanguageLevel 1, the corners of a halftone cell had to coincide with device pixels; this restriction resulted in actual screen angles being different from requested screen angles. A supercell contains multiple individual halftone cells. The corners of the supercell had to coincide with device pixels, but the corners of the individual halftone cells were no longer required to do so. The supercell mechanism provided for much more accurate screen angles.

Note *For more information on the use of halftones, halftone screens, and halftone cells, see Section 6.4 of the PostScript Language Reference Manual, Second Edition.*

1.4 Halftones in LanguageLevel 3

LanguageLevel 3 extends the halftone model to introduce new halftone types and capabilities. Five new halftone types have been added as well as extensions to the current definitions of halftone screens and halftone cells to include the use of a *supercell* (defined below). Section 2 covers the changes to halftone screens and halftone cells to provide more photorealistic gray levels in mid-range PostScript printers. Section 3 covers the additional halftone types, including Type 16, which provides more photorealistic gray levels in high-end PostScript printers and imagesetters.

Note *A new user parameter called* ***HalftoneMode*** *has been added to the PostScript language. The value of* ***HalftoneMode*** *affects the behavior of the halftone setting operators* ***setscreen****,* ***setcolorscreen****, and* ***sethalftone****. See Section 10.2 of the Supplement: PostScript Language Reference Manual, for more information.*

2 Supercells

Section 2.1, “An Overview of Halftone Screens and Supercells” covers the use of a new halftone supercell for creating up to 256 levels of gray in mid-range PostScript printers. Section 2.2, “Implementing Supercells with MaxSuperScreen” describes how this parameter is used to control the halftone supercell. Section 2.3, “The Benefits of Using Supercells” covers some of the benefits of using the new halftone supercell.

2.1 An Overview of Halftone Screens and Supercells

Even though the PostScript interpreter can handle 256 levels of gray, most PostScript printers can only print between 32 to 72 levels of gray.

In LanguageLevel 2, the operators **setscreen** and **setcolorscreen** and the **HalftoneType 1** halftone dictionary use only very small halftone cells with very few printable gray levels. This method can produce banding in blends and contouring in images such as faces or the sky.

Halftone cells are defined by a frequency and an angle. The number of printable levels of gray (*n*) can be estimated with the following formula:

$$n = (resolution/frequency)^2 + 1$$

Table 2 shows screens that are commonly used in desktop printers as the default screen or as default screens for individual color planes in a color printer. Using the above formula, it can be shown that none of these screens provides more than 100 levels of gray.

Table 2 *Common Screens Used in Monochrome and Color Printers*

Frequency	Angle	Resolution	Grays
53.03	45.000	300	33
47.43	18.435	300	41
50.00	0.000	300	37
70.71	45.000	600	73
63.25	18.435	600	91
66.67	0.000	600	82

In LanguageLevel 3, more gray values can be produced because all halftone screens defined by spot functions are implemented using a 2x2 *supercell* (a supercell is a grouping of four halftone cells in a 2x2 pattern). The use of the supercell increases the number of gray levels by a factor of four, to 256 levels of gray. These additional gray values allow for more photorealistic output.

Table 3 shows the same screens as above with the use of a halftone supercell. All of the screens now provide up to 256 levels of gray on *bilevel devices* (devices where each pixel is either black or white) and other halftone devices. A bilevel device is a subset of halftone devices.

Table 3 *Number of Gray Levels Available using Supercells*

Frequency	Angle	Resolution	Grays
53.03	45.000	300	129
47.43	18.435	300	161
50.00	0.000	300	145
70.71	45.000	600	256
63.25	18.435	600	256
66.67	0.000	600	256

2.2 Implementing Supercells with MaxSuperScreen

The implementation of photorealistic grays using supercells is controlled by the printer device user parameter, **MaxSuperScreen**, which sets an upper limit for the number of pixels in the halftone supercell.

The value 1016 (254 ∗ 4) is the highest *effective* value that **MaxSuperScreen** can have (although it can support higher values up to 1024). This takes place when 256 input values are printed on bilevel devices.

The following conditions must be met for a supercell to be created:

- The number of pixels in the supercell must be less than or equal to the value of **MaxSuperScreen**.
- The supercell must be within the limits of the values of the **MaxScreenItem** and **MaxScreenStorage** user parameters (see Appendix C of the *PostScript Language Reference Manual, Second Edition*).
- The number of pixels in the original halftone cell must be less than the number of usable thresholds.
- The halftone must be created with a spot function. This can be done with setscreen, setcolorscreen, or Type 1 halftones (or Type 5 halftones which contain Type 1 halftones).

If these conditions are not met, the supercell is not created and the original halftone cell is used instead. The last condition prevents the unnecessary use of supercells. If the original halftone cell provides all of the threshold values

that the renderer can use (or more than it can use), then a supercell adds no more printable levels of gray. The number of usable thresholds can be calculated with the following formula:

usable thresholds = (input levels – 1) / (output device levels – 1)

Table 4 shows the number of usable thresholds, by bits per component, and the resulting maximum usable supercell size.

Table 4 *Usable Thresholds and Resulting Maximum Usable Supercell Size*

Input Levels	Bits Per Component	Usable Thresholds	Maximum Usable Supercell Size
256	1	155	1016
256	2	85	336
256	4	17	64
256	8	1	0

The first three rows in Table 4 refer to halftone devices. The last row represents a contone device.

Note *For more information on the **MaxSuperScreen** user parameter, see Section 10.1 of the Supplement: PostScript Language Reference Manual, or the PostScript language Printer Addendum that accompanies each PostScript printer.*

2.3 The Benefits of Using Supercells

There are several benefits in using supercells for halftone screens:

- Moderate-resolution devices can now support up to 256 tone levels per color component instead of being limited to 32 (for 300 dpi devices) or 68 (for 600 dpi devices).
- Use of supercells is not computationally expensive and only requires a small amount of extra RAM.
- Supercells add better quality grays and performance increases on desktop devices. Blends that use between 100 and 256 gray levels will look smoother, with steps that are either less visible, or actually invisible, to the viewer.
- The use of supercells can contribute to better photorealistic color reproduction.
- The use of supercells results in smoother blends and gradients and higher overall quality on images printing to a low- to medium-resolution device.
- Supercells eliminate the need for printer manufacturers to create Type 3 halftones for this purpose.

3 Halftone Dictionaries

Section 3.1, “An Overview of the New Halftone Dictionaries” gives an overview of the additions and changes to the halftone dictionaries supported in LanguageLevel 3. Section 3.2, “The HalftoneType 16 Halftone Dictionary” provides a detailed look at the use of the new type 16 halftone dictionary for creating more than 256 levels of gray on high-end PostScript printers. Finally, Section 3.3, “The Benefits of Using Type 16 Halftones” presents some of the benefits of using type 16 halftones.

3.1 An Overview of the New Halftone Dictionaries

In LanguageLevel 3, a change has been made to all of the current halftone types. In addition, several new dictionary types have been added.

The **HalftoneName** key has been added to all halftone dictionaries, including types 1 through 5. This key is used by the **GetHalftoneName** operator that is part of the **ColorRendering ProcSet**.

Note *For more information on the **HalftoneName** key for halftone dictionary types **1** through **5**, see Section 6.3 of the Supplement: PostScript Language Reference Manual, for more information.*

Additionally, halftone types 1 through 5 are still required in all PostScript printers (they were required in LanguageLevel 2).

Note *For more information on halftone dictionary types 1 through 5, see Section 6.3 of the PostScript Language Reference Manual, Second Edition.*

The following is a list of halftone dictionaries that have been added to LanguageLevel 3:

- Type 6: is similar to a Type 3 halftone. A Type 6 halftone defines a threshold array at device resolution. This array is held in a file rather than a string, which allows for a larger threshold array size – larger than 64Kb. This halftone type is a standard feature in all PostScript interpreters. The keys for a Type 6 halftone dictionary are listed in the following table:

Key	**Type**	
HalftoneName	name or string	optional
HalftoneType	integer	required
Height	integer	required
Thresholds	file	required
TransferFunction	procedure	optional
Width	integer	required

Example 2 uses a type 6 halftone. It is similar to the type 3 halftone example shown earlier. In fact, the threshold array is the same, but the data comes from a file object (like **currentfile**) instead of being contained in a string within the halftone dictionary. After the halftone has been set, it is used in conjunction with a smooth shading dictionary and the **shfill** operator to draw a color gradient fill across a rectangle centered on the page.

Example 2 *Type 6 Halftone with Threshold Arrays in Files*

```
%!PS
/inch {72 mul} def
<<% start halftone dictionary
    /HalftoneType 6 % 8-bit threshold data from file
    /Width 20 % 20 pixel wide cell
    /Height 40 % 40 pixel high cell
    /Thresholdscurrentfile /ASCIIHexDecode filter
>> sethalftone
% Add threshold data here
58654B0B4EF98316ED092395D633B7297C641223
... % more data here
59D567EA1EACD159B8804EB9E237A18BF4106535
>

<<% start shading dictionary
    /ShadingType 2 % axial shading
    /ColorSpace /DeviceRGB % RGB color space
    /BBox [1 inch 1 inch 7.5 inch 10 inch]
    /Coords [4.25 inch 1 inch 4.25 inch 10 inch]


    /Function <<% start function dictionary
        /FunctionType 2 % exponential interpolation
        /Domain [0 1]
        /C0 [0 1 0.5]
        /C1 [1 0 1]
        /N 1
    >>% end function dictionary
>> shfill % smoothly shade the region
showpage
```

- Type 9: defines a halftone with proprietary data. This type has been added to support manufacturers that have special halftone requests or requirements. The details are built into the system and can not be interrogated by an executing PostScript program. This halftone type is optional for Postscript printers. The keys for a Type 9 halftone dictionary are listed in the following table:

Key	Type	
HalftoneName	name or string	optional
HalftoneType	integer	required

Note *Most printers do not support Type 9 halftones. This type of halftone should normally not appear in a PostScript language program. That is, it should never be specified by an application.*

- Type 10: specifies a threshold array that represents a halftone cell with a non-zero screen angle. A simple transformation is applied to the halftone cell converting into two squares, making it easier to specify non-zero angle cells. This halftone type is a standard feature for all PostScript printers. The keys for a Type 10 halftone dictionary are listed in the following table:

Key	Type	
HalftoneName	name or string	optional
HalftoneType	integer	required
Xsquare	integer	required
Ysquare	integer	required
Thresholds	string or file	required
TransferFunction	procedure	optional

- Type 100: specifies a halftone similar to a Type 9 with the optional of additional, manufacturer-defined keys. This halftone type is optional for all PostScript printers. The keys for a Type 100 halftone dictionary are listed in the following table:

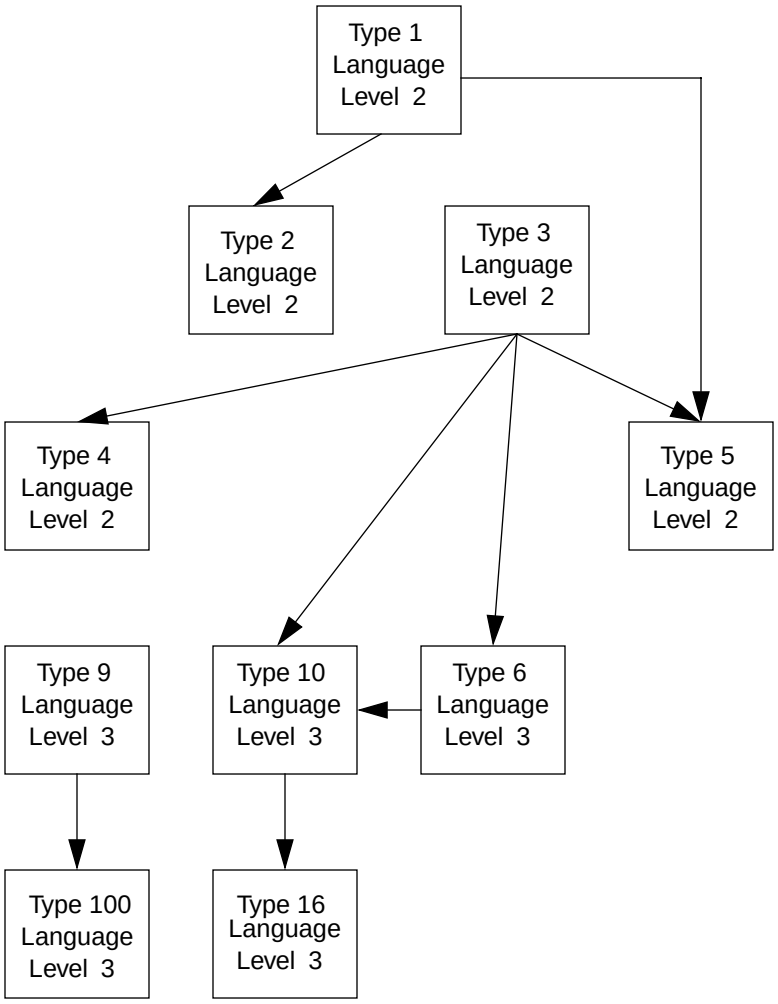
Key	Type	
HalftoneName	name or string	optional
HalftoneType	integer	required
other	any	optional

Note *Most printers do not support Type 100 halftones. This type of halftone should normally not appear in a PostScript language program. Like the Type 9 halftone, this halftone type should not be used by an application. There would be no way to predict the unknown parameters specified by a printer manufacturer.*

Note *For more information on halftone types **6**, **9**, **10**, and **100**, see Section 6.3 in the Supplement: PostScript Language Reference Manual.*

Figure 1 shows the relationship of all of the halftone types supported in LanguageLevel 3.

Figure 1 *The Family Tree of Halftone Dictionary Types*



Note that in Figure 1, Type 6, Type 9, Type 10, Type 16, and Type 100 halftones can all be inputs to Type 5 (this was already true for Type 1 and Type 3).

Note *There are five new instances of the implicit resource **HalftoneType**. These are types **6**, **9**, **10**, **16**, and **100**. See Section 3.1 of the Supplement: PostScript Language Reference Manual. for more information.*

3.2 The HalftoneType 16 Halftone Dictionary

The Type 16 halftone is defined similarly to the Type 10 and Type 6 halftones. In both cases, a halftone screen is defined directly with a threshold array that is specified at device resolution. For Type 16 halftones, each element of the threshold array is defined as 16 bits, rather than 8. This change allows for the specification of 2^{16} (or 65,536) levels of gray/color rather than the 256 levels that could be achieved with a Type 6 halftone. Thus, with Type 16 halftones, it is possible to render grays/colors more accurately, and color gradient fills more smoothly.

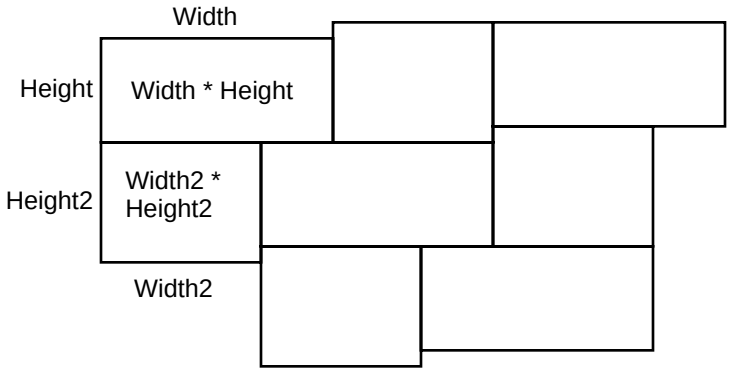
Note *The Type 16 halftone is only useful for imagesetter-type devices. Other types of printing devices will not benefit from the use of this halftone type.*

Note *In the current version of the PostScript interpreter, this additional accuracy is only available with the smooth shading feature. When rendering any graphical element other than a smooth shading, only 8 bits of accuracy are used. The additional accuracy in not available on all products, and not on devices with other than 1 bit per component.*

For Type 16 halftones, the threshold array can be defined by either one (*Width* ∗ *Height*) or two (*Width2* ∗ *Height2*) rectangles.

Figure 2 shows how the device space is tiled when two rectangles are defined. The last row in the first rectangle is immediately adjacent to the first row in the second rectangle, and the rows start in the same column.

Figure 2 *Tiling the Device Space in a HalftoneType 16 Halftone Dictionary*



The **sethalftone** operator can be used to install a Type 16 halftone dictionary. For this type of halftone, the first *Width* ∗ *Height* ∗ *2* characters are read from the **Thresholds** file (see Table 5 below); if a second rectangle is defined, then another *Width2* ∗ *Height2* ∗ *2* characters are read. The resulting threshold array is saved in internal storage. A **rangecheck** error will result if not enough data is available to define the two rectangles. If no error occurs, the file is closed on EOF; otherwise, it is left open.

The **currenthalftone** operator returns a halftone dictionary whose **Thresholds** file points to the current threshold array in internal storage. This file object will be different than the one given to the **sethalftone** operator. In addition, the file is nonreadable and nonaccessible. The **sethalftone** operator does recognize it as the threshold array stored internally.

Table 5 lists the keys used in a HalftoneType 16 halftone dictionary.

Table 5 *Keys for HalftoneType 16 Halftone Dictionaries*

Key	Type	
HalftoneName	name or string	optional
HalftoneType	integer	required
Height	integer	required
Height2	integer	optional
Thresholds	file	required
TransferFunction	procedure	optional
Width	integer	required
Width2	integer	optional

HalftoneType must be set to 16.

HalftoneName, if present, supplies the name of the halftone dictionary to the **findcolorrendering** operator that forms the name of a color rendering dictionary (CRD). The **HalftoneName** key is also used by the **GetHalftoneName** operator that is part of the **ColorRendering ProcSet**.

Height specifies the height, in pixels, of the first (or only) rectangle in the threshold array.

Width specifies the width, in pixels, of the first (or only) rectangle in the threshold array.

Height2 specifies the height, in pixels, of the second rectangle in the threshold array. If this key is specified, **Width2** must also be specified (that is, both are required to define the second rectangle of the threshold array); if **Height2** is omitted, **Width2** can not be specified and there is no second rectangle defined for the threshold array.

Width2 specifies the width, in pixels, of the second rectangle in the threshold array. Both **Width 2** and **Height2** are required keys to define the second rectangle of the threshold array.

Thresholds is a file that must hold (*Width* ∗ *Height* ∗ *2*) characters or ((*Width* ∗ *Height* ∗ *2*) + (*Width2* ∗ *Height2* ∗ *2*)) characters of threshold data. Each threshold is 16 bits wide. The high-order character of each threshold is stored first.

Note *This file object can be returned by the* ***currentfile*** *operator.*

If **TransferFunction** is present in the dictionary, it overrides the transfer function normally specified by the **settransfer** or **setcolortransfer** operator.

Note *Type 16 halftones are a standard feature for all PostScript printers.*

Example 3 illustrates the use of Type 16 halftones in conjunction with a smooth shading operation. Just as with Type 6 halftones, the threshold array for a Type 16 halftone comes from a file. The difference is that each element of the threshold array is sixteen bits in a Type 16 halftone instead of eight bits as in a Type 6 halftone. This example illustrates a Type 16 halftone that contains only one halftone tile defined in the threshold array.

Example 3 *Type 16 Halftone with Threshold Arrays in Files*

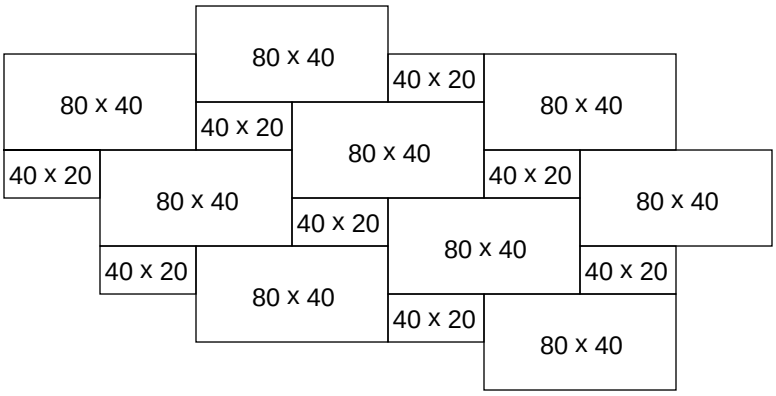
```
%!PS
/inch {72 mul} def
<<% start halftone dictionary
    /HalftoneType 16 % sixteen-bit data from file
    /Width 20 % 20-pixel wide cell
    /Height 40 % 40-pixel high cell
% The sethalftone operator will read 20 x 40 x 2
% characters from the data source
/Thresholds currentfile /ASCIIHexDecode filter
>> sethalftone
585865654B4B0B0B4E4EF9F983831616EDED090923239595D6D633
... % continue threshold data here
8B8BF4F4101065653535
>
<<% start shading dictionary
    /ShadingType 2 % axial shading
    /ColorSpace /DeviceGray % RGB color space
    /BBox [1 inch 1 inch 7.5 inch 10 inch]
    /Coords [4.25 inch 1 inch 4.25 inch 10 inch]
    /Function <<% start function dictionary
        /FunctionType 2 % exponential interpolation
        /Domain [0 1]
        /C0 [0]
        /C1 [1]
        /N 1
    >>% end function dictionary
>> shfill % smoothly shade the region
showpage
```

Example 4 shows a Type 16 halftone dictionary that describes two halftone tiles represented by two threshold arrays. The first halftone tile is 80 pixels wide and 40 pixels high. The second halftone tile is half the size of the first, namely, 40 pixels wide and 20 pixels high. When the halftone machinery is in action, the tiles are laid down on the pixel grid as shown in Figure 3.

Example 4 *Type 16 Halftone with Two Threshold Arrays in Files*

```
%!PS
/inch {72 mul} def
<<% start halftone dictionary
    /HalftoneType 16 % 16-bit threshold data from file
    /Width 80 % 80 pixels wide in first tile
    /Height 40 % 40 pixels high in first tile
    /Width2 40 % 40 pixels wide in second tile
    /Height2 20 % 20 pixels wide in second tile
    /Thresholds currentfile /ASCIIHexDecode filter
>> sethalftone
% Threshold data goes here
1A7EDEF4E6FE2DF4B365683DC1A51A080CD6775EEC1B1CE67A813A8BF7A8B
9A2605B15F890A9
... % continue data
503CC306C7011C780CA9D9DD8F131798B8ED0472
>
<<% start shading dictionary
    /ShadingType 2 % axial shading
    /ColorSpace /DeviceGray % Gray color space
    /BBox [1 inch 1 inch 7.5 inch 10 inch]
    /Coords [4.25 inch 1 inch 4.25 inch 10 inch]
    /Function <<% start function dictionary
        /FunctionType 2 % exponential interpolation
        /Domain [0 1]
        /C0 0
        /C1 1
        /N 1
    >>% end function dictionary
>> shfill % smoothly shade the region
showpage
```

Figure 3 *Tiling Resulting from Threshold Arrays in Example 4*



3.3 The Benefits of Using Type 16 Halftones

There are several benefits of Type 16 halftones:

- The addition of the Type 16 halftone allows high-resolution or high-precision devices, such as imagesetters, that were previously limited to 256 tone levels per color component to support up to 4096 (2^{12}) levels.

- It is now possible to render colors more accurately and color gradient fills more smoothly when performing smooth shading on devices that have sufficient output resolution. This can contribute to more and/or better photorealistic color reproduction.

4 Tips and Techniques

4.1 Best Uses for Supercells

Supercells can be used on PostScript devices such as monochrome, color, ink-jet, and laser printers, and imagesetters. They are best used on devices that output 1 or 2 bits per color component. They have limited value for devices with 4 bits per color component. They have no value for devices with 8 bits per color component (since these are considered continuous tone devices and do not use halftoning).

4.2 Recommendations for Setting MaxSuperScreen

It is recommended that the value of **MaxSuperScreen** be 1024. This is the Adobe standard value for this parameter.

4.3 Recommendations for Acquiring and Using Pre-Defined Threshold Arrays

Obtaining a correctly matched set of screens for a given device is not a trivial task. The actual process is conceptually simple: the desired screen is generate at a chosen frequency, angle, and spot function; the data that the spot function generates is collected; this data us used to build the threshold array.

In practice, the expertise to create these screens lives within a few companies that are in the high-end imagesetter and prepress business. There are also printer manufacturers and other companies that either build threshold arrays or supply applications that build threshold arrays.

Several printer manufacturers have generated their own threshold arrays. The arrays are usually implemented with Type 9 or Type 100 halftones. This approach was taken for Adobe® Brilliant Screens™.

4.4 Recommendations for using Type 16 halftones

Type 16 halftones are best suited for devices with a actual resolution above 1000 dpi (such as imagesetters).

With the current release of the PostScript language, Type 16 halftones are used only in conjunction with the smooth shading machinery. When used in any other context, only 8 bit accuracy can be achieved. Therefore, it is recommended that Type 16 halftones only be used for smooth shading on high-end PostScript printing systems.

Adobe Systems Incorporated

Index

T

W