Connectivity of the HP DeskJet 1200C Printer

The connectivity components include the language firmware, a language interface to the mechanical firmware, software printer drivers, and tools for various environments and for driver developers. A screen calibrator tool enlists the user's help in making the printed output match the screen.

by Anthony D. Parkhurst, Ramchandran Padmanabhan, Steven D. Mueller, and Kirt A. Winter

"Connectivity" is the term used to refer collectively to the software and firmware components of the HP DeskJet 1200C printer that, along with the printer itself, bring to the user a complete color printer solution. These components are the language firmware, a language interface to the mechanical firmware, software printer drivers and other tools for the Microsoft® Windows and Macintosh environments, and software tools for MS-DOS® driver developers.

The language, PCL 5C, is a colorized and (continually) enhanced version of the highly efficient printer command language developed internally by HP to provide consistent feature support across its printer product lines, from the DeskJet to the PaintJet to the LaserJet lines. The language system interface to the mechanical firmware is a well-defined interface that allows outside language firmware developers, such as Adobe Systems, to develop other language processes besides PCL, such as PostScript,™ for the printer. HP printer drivers are written for the Windows and Macintosh environments, and include highly advanced color management tools that allow the user to achieve the best color quality available in the DeskJet 1200C. These color management tools are also available to independent software vendors developing DOS drivers for the printer. Topics discussed in this article are:

- The PCL 5C language firmware
- Raster operations, a PCL 5C language feature that provides advanced drawing functionality for the printer driver
- The language interface to the mechanical firmware, hereafter called the language interface
- A screen calibration tool for color management.

PCL 5C Language Firmware

The HP DeskJet 1200C printer unifies HP LaserJet printer intelligence with high-quality color printing by combining a new generation of inkjet technology with a color enhanced version of the LaserJet language known as PCL 5C. PCL 5C gives applications access to state-of-the-art font scaling technologies (Intellifont and TrueType™), full-featured vector graphics with the HP-GL 2 language, and 24-bit color imaging.

The programming, or firmware, in the DeskJet 1200C contains a PCL 5C page formatter subsystem and a subsystem to control the hardware and electronics. The PCL 5C page formatter converts the high-level data stream from the computer into a simple data stream for the print engine. This interpreter consists of two processes: the parser and the image processor.

Parser

The parser recognizes commands in the data stream and performs the requested functions. The output of the parser is a *display list* that is sent to the image processor. PCL 5C is a *page description language*: the data stream describes what the page should look like and prints the page when it is complete. Page description consists of objects combined with colors and patterns. An obvious benefit of a page description language is that the objects do not have to be created in a position dependent order (e.g., top to bottom), so the host need not merge or reorder objects before sending them. The kinds of objects used in PCL 5C are text, rules (rectangles), HP-GL 2 graphics (vectors and polygons), and images (see Fig. 1).

After specifying character position on the page, the typeface desired, and the point size (72 points/inch), the application sends character codes to create text. The parser generates a bitmap of the requested character from a scalable typeface and stores the bitmap into the display list with the appropriate positioning information. If this identical character is requested later in the job (which is common for English text), the bitmap is reused. PCL 5C can scale characters from ¼ point to 999¾ points with astonishing quality.

Rules, which are simply rectangles, are stored on the display list as a starting coordinate pair along with a width and a height.

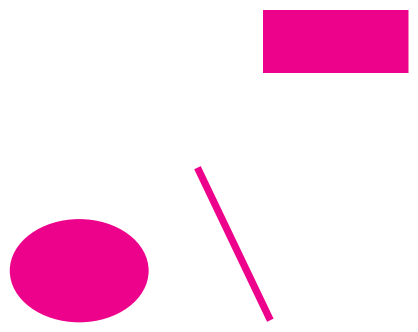


Fig. 1. Basic object types: text, rules, polygons, vectors, and raster.

HP-GL 2, the vector graphics language for pen and large-format plotters, is a subparser within PCL 5C. HP-GL 2 has features that provide high-level graphics, including polygons. All HP-GL 2 objects are scaled, clipped, and decomposed into simple vectors and trapezoids to be stored on the display list. An additional feature of HP-GL 2 within PCL 5C is the ability to use outlines from the font scaling technology directly as polygons. This gives the application the ability to create special effects with text.

All of these objects are basically colorless—on a monochrome printer such as a LaserJet, they will print as black. The parser treats them as black objects. Color is handled independently and is not applied to these objects until they get to the image processor. To work with color, the application must first configure a color palette. For compatibility with monochrome printers, the default palette is defined as having two colors: black and white. Palette configuration is performed in three steps. First, the size and color space of the palette are defined. Next, each entry of the palette is assigned a color in that color space (up to 24 bits of color information per entry). Finally, color can be adjusted for gamma correction and rendering algorithm. Color can be used as foreground color for objects such as text and vectors, in color patterns, and in raster images. When such a color is selected, the parser stores the color on the display list, and the image processor will apply this color to each of the objects on the display list until another color is selected. Another method of using color is to define a color pattern to apply to the objects. A color pattern differs from foreground color in that several colors can be specified within the pattern. The parser stores this pattern on the display list for the image processor to apply to objects. The third and final way to use color is raster images.

Raster processing in PCL 5C is very flexible, but most applications send raster data in one of four ways. Images can be sent as single-plane bitmaps (monochrome), which are LaserJet-compatible. Color can be applied to these bitmaps in the same way as other black objects. Another form of raster is 3-plane simple color. This mode is for applications that do most of the work themselves and prefer to simply send a color bitmap. Each pixel can take on eight values to represent the colors that the print engine can print (black, red, green, blue, cyan, magenta, yellow, and white). For higher-level applications, data is represented as 8 bits per pixel, indexed. Each pixel can have a value from 0 to 255, and that value is used to index into a palette where each entry is a color defined in a 24-bit color space. Finally, for the highest-quality images from color scanners, data can be sent directly as 24 bits per pixel. Palette entries are not used for this mode. Standard techniques (dithering and error diffusion) are used to reduce the continuous-tone image to 3-plane simple color for the print engine. The resulting image is stored in the display list.

The Image Processor

The image processor takes objects from the display list, applies colors or patterns and puts them in *swaths*. On this printer, swaths consist of three bitmaps, one each for cyan, magenta, and yellow. Swaths are 2464 dots wide by 128 dots high and are used by the print engine to drive the inkjet printheads. The width of the swath is determined by the printable area of the largest supported media size, and the height is established by performance and efficiency issues. Although text and raster are fully formed bitmaps at this point, rules, HP-GL 2 vectors, and trapezoids are simply coordinate pairs. The image processor must create bitmaps for these objects. Rules are simple, since they are rectangles that lie horizontally or vertically, but vectors and trapezoids are not so well-behaved. There are well-established vector-to-raster conversion algorithms, but the DeskJet 1200C uses a high-performance vector-to-raster conversion process developed for HP large-format electrostatic plotters.[1] This method uses precomputed bitmap tiles stored in ROM to build vectors (see Fig. 2). Tiles are bitmaps that are aligned to word boundaries and can be pasted as needed in the swath. The image processor selects the tiles based on slope, width and position of the vector relative to word (16-bit) boundaries.

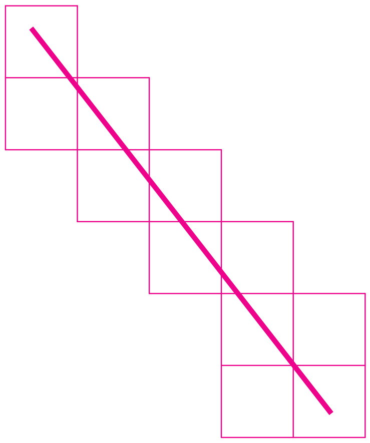


Fig. 2. Precomputed tiles are used to create a vector.

At this point, all the objects (text, rules, tiles, images) look like bitmaps. To complete this architecture, colors and patterns on the display list are also tile bitmaps. Similar to vector-to-raster tiles, these tiles can be pasted anywhere in the swath along word boundaries. When a foreground color is selected, the parser gets the color value from the palette, then applies the current dither matrix to create a 16-by-16-bit color tile (the size of the dither matrix is 16 by 16 bits) which is stored on the display list as a three-plane object: one plane each for cyan, magenta, and yellow. When the image processor needs to apply a color tile to an object, the color tile is logically replicated to cover the page, then combined or masked with the object, and the result is put in the swath (Fig. 3). A monochrome pattern in effect at this time will also be combined with the object. A color pattern differs from a monochrome pattern in that the color pattern already has color applied to it. The logic used in combining color tiles, pattern tiles, object bitmaps, and the destination bitmap (swath) is selected by the application and explained in “Raster Operations” below. When all the objects for each swath are processed, the swath is printed.

An alternative to the display list would be a frame buffer, which is a full-page bitmap used to create the final output. Although this approach has its advantages (especially with very complex graphics), it consumes far more memory than most jobs require. The display list’s efficient use of memory allows the printer firmware to work farther ahead of the print engine, providing smooth and rapid printing.

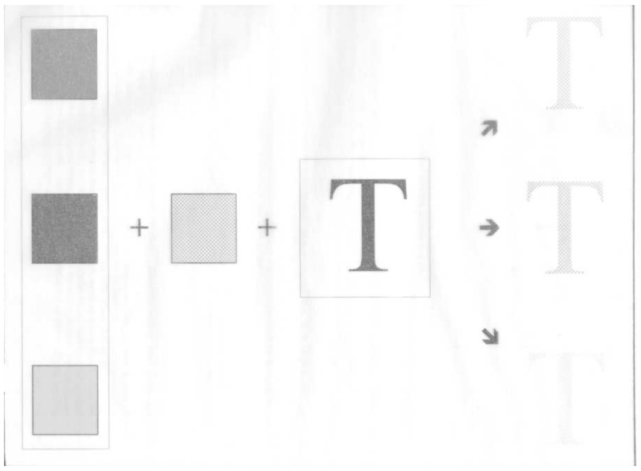


Fig. 3. The cyan, magenta, and yellow color components are combined with the pattern and bitmap.

Is all this functionality worth the extra product cost to provide it? Yes. With a high-level language like PCL 5C, the printer can do much of the work that the host computer would have to do for a bare raster printer. This allows the host computer to service the user instead of the printer. Because the DeskJet 1200C is LaserJet-compatible, LaserJet applications continue to work for the DeskJet 1200C, and driver writers simply have to add color functionality to their LaserJet drivers instead of having to create yet another printer driver from scratch.

Raster Operations

The world of graphical output devices is divided into two distinct categories: raster devices and vector devices. Most color and laser printers are raster devices, although graphic applications communicate with them through high-level page description languages. This provides a level of abstraction away from hardware, allowing the applications to specify attributes and lay down geometrical shapes in a device independent fashion. Although the device uses pixels for its final graphic representation, applications do not need to talk to the interface in terms of pixels. For instance, an application can take advantage of the fact that the printer can draw circles by just sending down a radius and the coordinate position of the center instead of every point on the circumference. The application can also specify a fill pattern for the object and lay it along with or on top of similar objects, blending their corresponding textures in several different ways.

Microsoft's Graphical Device Interface (GDI) is a suite of interface specifications that bridges the gap between various Windows-based GUI (graphical user interface) applications and a wide range of graphical output devices. GDI is a high-level interface to a graphical output device. The HP DeskJet 1200C driver for Microsoft Windows applications can directly map several GDI primitives and drawing modes to PCL 5C and HP-GL 2 high-level object descriptions. A commonly used parameter in many GDI primitives is called a logical operation or a *raster operation* (ROP).

ROPs are drawing modes that define how the bitwise interaction between a high-level source object, a color texture, and the current destination produces a new destination. Examples of source objects are vectors, polygons, text, and raster images. Texture is a color or monochrome mask that is applied to the source in a manner specified by the ROP. Some applications have objects whose color is based not only on the currently selected foreground color, but also on the original contents of the destination.

Theory of ROPs

ROPs directly affect the appearance of objects drawn on the display device by making use of both the selected foreground color and the current contents of the destination. For instance, an object can be placed on top of another object so that it either completely overlaps its destination area or its colors merge with those of the underlying object. The drawing mode specified by the ROP makes it possible to choose one of the several alternatives for rendering the object by defining a bitwise Boolean operation between the pixels of the source and the pixels of the destination.

Consider a monochrome device in which an object can be drawn only with a black print cartridge on a destination whose pixels are either black or white. There are four ways of combining a black or white source pixel with a black or white destination pixel.

Source	1 1 0 0
Destination	1 0 1 0

where 0 is black and 1 is white.

A bitwise Boolean operation for each of the above four cases can yield either a 0 or a 1 in each of the four bit positions. This results in 16 two-operand ROPs. These are the monochrome drawing modes. A complete list of monochrome drawing modes is shown in Table I.

Table I
Monochrome Drawing Modes

Source (S) / Destination (D)		1 1 0 0 / 1 0 1 0	Monochrome Drawing Mode	
Result	0	0 0 0 0	0	BLACK
	1	0 0 0 1	~(S | D)	NOT_MERGE_PEN
	2	0 0 1 0	~S & D	MASK_NOT_PEN
	3	0 0 1 1	~S	NOT_COPY_PEN
	4	0 1 0 0	S & ~D	MASK_PEN_NOT
	5	0 1 0 1	~D	NOT
	6	0 1 1 0	S ^ D	XOR_PEN
	7	0 1 1 1	~(S & D)	NOT_MASK_PEN
	8	1 0 0 0	S & D	MASK_PEN
	9	1 0 0 1	~(S ^ D)	NOT_XOR_PEN
	10	1 0 1 0	D	NOP
	11	1 0 1 1	~S | D	MERGE_NOT_PEN
	12	1 1 0 0	S	COPY_PEN
	13	1 1 0 1	S | ~D	MERGE_PEN_NOT
	14	1 1 1 0	S | D	MERGE_PEN
	15	1 1 1 1	1	WHITE

The symbols ~, ^, &, and | represent the bitwise logical operations NOT, exclusive-OR (XOR), AND, and OR, respectively.

Color Drawing Modes

When color is introduced into the device, both the source and the destination have three planes of data representing eight primary colors: the additive colors red, green, and blue, the subtractive colors cyan, magenta, and yellow, and black and white. Here the monochrome ROPs are performed separately for each of the three planes to arrive at the consolidated resultant destination.

At any level of drawing, solid geometrical objects are drawn as outlines and filled with a color pattern using a brush. It is no different in computer graphic art packages. A color pattern generated by a brush is called a texture. Textures are normally masked through solid objects and placed in their final destinations. The drawing modes must support an additional parameter because there is more than one way to make use of a texture brush. The color drawing modes consist of a set of three-operand ROPs that combine the pixels of source, texture, and destination to produce new destination pixels. There are eight ways of grouping a Boolean source, texture, and destination pixel on a single plane:

Texture	1 1 1 1 0 0 0 0
Source	1 1 0 0 1 1 0 0
Destination	1 0 1 0 1 0 0 0

Extrapolating our derivation of monochrome drawing modes to the color case results in 256 possible ways of combining source, texture, and destination to produce a new destination. ROPs on color data are performed on a plane-by-plane basis. Some of the most commonly used ROPs are shown in the following table.

Table II
Color Drawing Modes in RGB Space

			Color Drawing Mode	
Texture (T)		**1 1 1 1 0 0 0 0**		
Source (S)		**1 1 0 0 1 1 0 0**		
Destination (D)		**1 0 1 0 1 0 1 0**		
Result	0	0 0 0 0 0 0 0 0	0	DARKNESS
	51	0 0 1 1 0 0 1 1	~S	NOT_SOURCE_COPY
	85	0 1 0 1 0 1 0 1	~D	DEST_INVERT
	90	0 1 0 1 1 0 1 0	T ^ D	TEXTURE_INVERT
	102	0 1 1 0 0 1 1 0	S ^ D	SOURCE_INVERT
	136	1 0 0 0 1 0 0 0	S & D	SOURCE_AND
	168	1 0 1 0 1 0 0 0	(T|S)&D	
	192	1 1 0 0 0 0 0 0	T & S	MERGE_COPY
	204	1 1 0 0 1 1 0 0	S	SOURCE_COPY
	234	1 1 1 0 1 0 1 0	(T&S)|D	
	240	1 1 1 1 0 0 0 0	T	TEXTURE_COPY
	252	1 1 1 1 1 1 0 0	T | S	
	255	1 1 1 1 1 1 1 1	1	BRIGHTNESS

A complete list of the 256 color drawing modes can be found in the programmer's reference manual of the Microsoft Windows software development kit and also in the HP DeskJet 1200C developer's guide.

Tables I and II show ROPs defined in RGB color space, whereas the DeskJet 1200C printer operates in a CMY color space (using cyan, magenta, and yellow print cartridges). The image processing firmware in the printer renders objects in CMY space and generates C, M, and Y planes of destination bitmaps. Users, however, being more used to the screen-oriented RGB model, send down RGB data that defines ROPs in RGB color space. The ROPs, therefore, have to be translated into CMY space for the image processor. Since RGB and CMY data complement each other, they can be converted easily from one to another. The ROP is transposed by toggling and swapping the bits within the byte so as to preserve the interpretation of the corresponding bitwise logical operation. ROPs defined in Table II are in RGB color space. Table III provides the CMY translation.

Table III
Color Drawing Modes in CMY Space

			Color Drawing Mode	
Texture (T)		**1 1 1 1 0 0 0 0**		
Pen Source (S)		**1 1 0 0 1 1 0 0**		
Destination (D)		**1 0 1 0 1 0 1 0**		
Result	255	1 1 1 1 1 1 1 1	1	DARKNESS
	51	0 0 1 1 0 0 1 1	~S	
	85	0 1 0 1 0 1 0 1	~D	
	165	1 0 1 0 0 1 0 1	~(T ^ D)	
	153	1 0 0 1 1 0 0 1	~(S ^ D)	
	238	1 1 1 0 1 1 1 0	S | D	
	234	1 1 1 0 1 0 1 0	(T&S)|D	
	252	1 1 1 1 1 1 0 0	T | S	
	204	1 1 0 0 1 1 0 0	S	
	168	1 0 1 0 1 0 0 0	(T|S)&D	
	240	1 1 1 1 0 0 0 0	T	
	192	1 1 0 0 0 0 0 0	T & S	
	0	0 0 0 0 0 0 0 0	0	BRIGHTNESS

ROPs and the PCL Print Model

The original PCL 5 print model specifies how to fill source image objects with any of the printer's predefined, shaded, hatched, or user-downloaded patterns. In addition to source, pattern, and destination, source and pattern transparency modes must be considered when masking the source image through the pattern tile onto the destination. Source transparency mode specifies whether the white pixels of the source image are transparent or opaque. Pattern transparency provides a similar definition for the white pixels of the pattern. Depending on the transparency modes, the destination either shows through the transparent pixels or gets blocked by the corresponding white pixels. PCL 5C also has a foreground color definition that gets applied to a monochrome pattern to generate what is known as a texture. A texture is either a color pattern or a colorized monochrome pattern (a monochrome pattern with foreground color applied to it).

PCL 5C implements a new print model, which performs transparency mode operations before it does ROPs.

ROP Formulas

We shall now proceed to derive the formulas used by the image processor to render objects into three-plane CMY bitmaps.

The main entities used are defined as follows:

Operands

Source:	Character cell, rules, color and monochrome raster images, HP-GL 2 vectors and polygons
Texture:	Either color pattern or (foreground color & pattern mask)
Destination:	Bitmaps for the C, M, and Y planes.

Operators

Source transparency and pattern transparency definitions, which generate a transparency mask

Color drawing modes (i.e., ROPs), which operate on the above three operands.

Notation

~Source denotes the white pixels of the source.

~Pattern denotes the white pixels of the pattern.

In a color source image, ~Source would be the white extract from the three planes of consolidated source data.

The generic formula used by the image processing firmware is based on the following premises:

- The destination shows through the white areas of the source if source transparency is on.
- The destination shows through the white areas of the pattern in the source area if pattern transparency is on.
- For all nonwhite pixels, the ROP result of the texture, source, and destination is the new destination.

Destination = (Destination & Transparency_Mask) | (ROP Result & ~(Transparency_Mask)),

where Transparency_Mask has to be defined for the four basic interactions of source and pattern transparencies with ROPs.

For the sake of simplicity, the default ROP has been chosen in Figs. 4, 5, 6, and 7. It is defined as (Texture OR Source) and converted to (Texture AND Source) in CMY space. These illustrations follow the CMY nomenclature.

Source Opaque and Pattern Opaque (Fig. 4). At present, this is the most commonly used model because most of the Windows GDI applications do not use the PCL transparency modes. Source and pattern transparencies are set to opaque and the transparency functionality is achieved using ROPs.

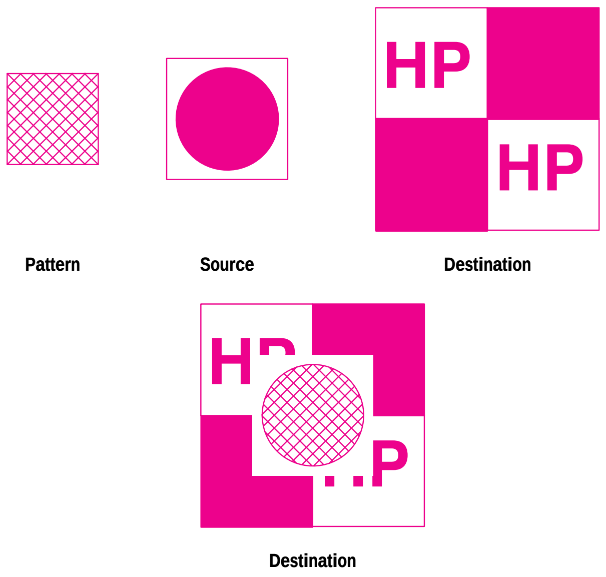


Fig. 4. The source opaque and pattern opaque transparency mode for the default raster operation.

Transparency_Mask = 0

Destination = ROP(Texture, Source, Destination)

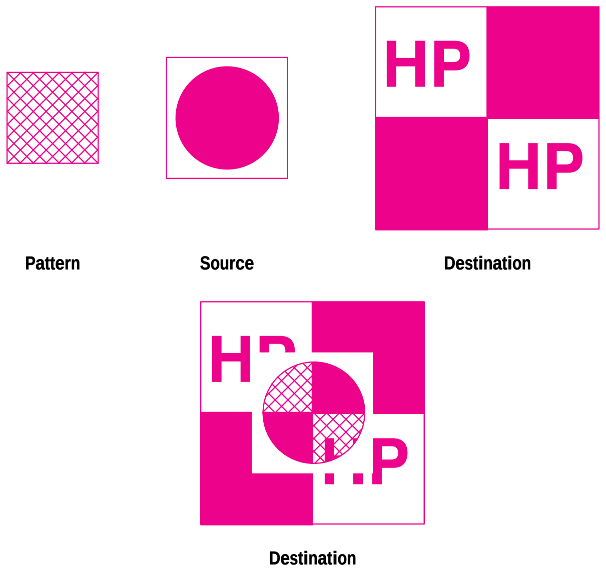


Fig. 5. The source opaque and pattern transparent transparency mode for the default raster operation.

Source Opaque and Pattern Transparent (Fig. 5). Since the pattern is transparent, the destination shows through the white areas of the pattern where the corresponding source pixels are nonwhite. However, the white area of the source is written onto the destination because the source is opaque. The result of the ROP is applied to the source pixel area of the destination, allowing the destination to show through the transparent source pixel area of the pattern.

Transparency_Mask = (Source & ~Pattern)

Destination = (Destination & (Source & ~Pattern)) | (ROP(Texture, Source, Destination) & ~(Source & ~Pattern))

Source Transparent and Pattern Opaque (Fig. 6). The destination shows through the white areas of the source mask. The source pixel area of the destination is cleared and the result of the ROP is laid down.

Transparency_Mask = ~Source

Destination = (Destination & ~Source) | (ROP(Texture, Source, Destination) & Source)

Source Transparent and Pattern Transparent (Fig. 7). This is the default case in the PCL 5C print model, and therefore the default case in the printer. The destination shows through the transparent area of the source. The destination also shows through the transparent area of the pattern tiled across the source. The result of the ROP is applied to the intersection of the source and the pattern. This ensures that the transparent areas of the source and pattern do not affect the destination.

Transparency_Mask = ~(Source & Pattern)

Destination = (Destination & ~(Source & Pattern)) | (ROP(Texture, Source, Destination) & (Source & Pattern))

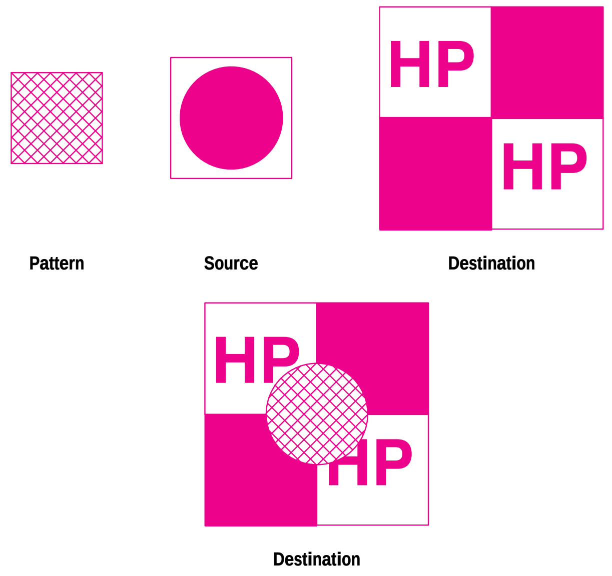


Fig. 6. The source transparent and pattern opaque transparency mode for the default raster operation.

Applications of ROPs

One typical application of ROPs is to clip objects to arbitrary clip paths. Consider an example of drawing an arbitrarily shaped object with a radial gradient texture. Fig. 8 shows the various steps taken to achieve this.

First the application sends down several concentric rings, each with a different shade of color. These are applied to the destination using ROP 90 (Destination = Destination XOR Texture), resulting in a circular gradient texture as shown in Fig. 8a.

The second step places a solid black arbitrarily shaped object (Fig. 8b) on top of the gradient using ROP 204 (Destination = Source). This results in Fig. 8c.

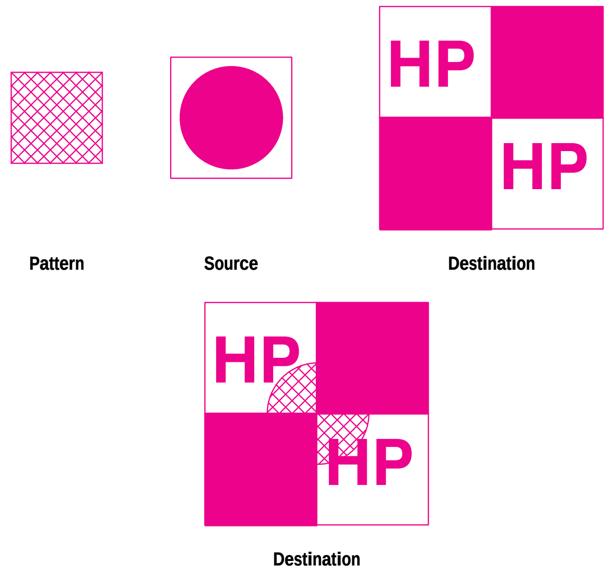


Fig. 7. The source transparent and pattern transparent transparency mode for the default raster operation.

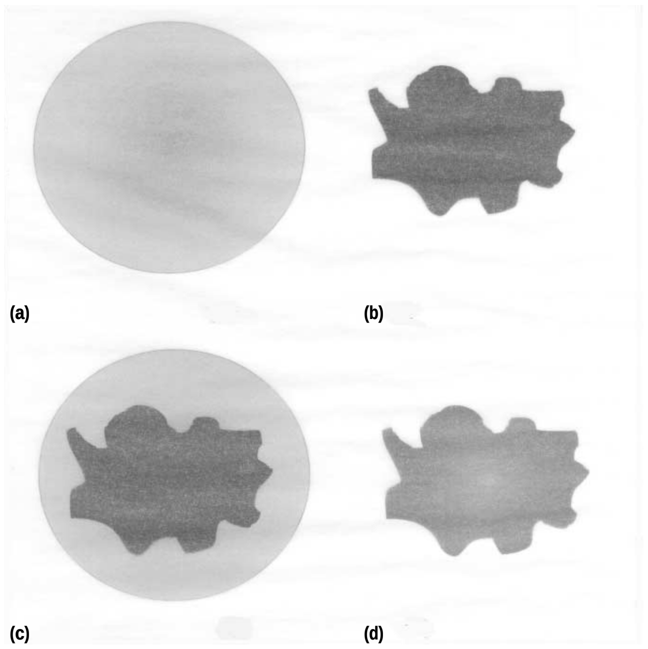


Fig. 8. Using raster operations to draw an arbitrarily shaped object with a radial gradient texture.

The final step sends the complete set of concentric rings all over again using ROP 90. This serves to erase the gradient pixels external to the object, resulting in the required radial gradient shading clipped to the arbitrary shape as in Fig. 8d.

Fig. 9 shows the result of a similar operation involving gradient-shaded text objects.

Pixel Placement Precision

Color space and address space are two fundamental differences between display devices and hard-copy devices. RGB-to-CMY conversion between screen-oriented applications and color printers has already been mentioned earlier in this article. Another area of incompatibility is the location of a pixel in the coordinate system. The printer places pixels at the intersections of the squares of a theoretical, device dependent grid covering the printable area of the page, whereas CRTs place pixels at the centers of the squares. These two pixel placement methods are called the *grid-intersection* method and the *grid-centered* method. They are shown in Figs. 10a and 10b, respectively.

A problem arises in the grid-intersection model when two adjacent polygons are defined and the ROP chosen performs an exclusive-OR operation on the polygons and the destination. The pixels along the border may be rendered twice, resulting in stray marks of a different color along the polygon edges. This looks very bad on the printed output because applications normally define complex graphical images using several little polygons of different shades of color.

The PCL 5C implementation in the DeskJet 1200C printer provides a choice of pixel placement methods. The default grid-intersection model places pixels at the intersections of the device dependent grid. A new grid-centered model simulates placing the pixel at the center of the square in the grid

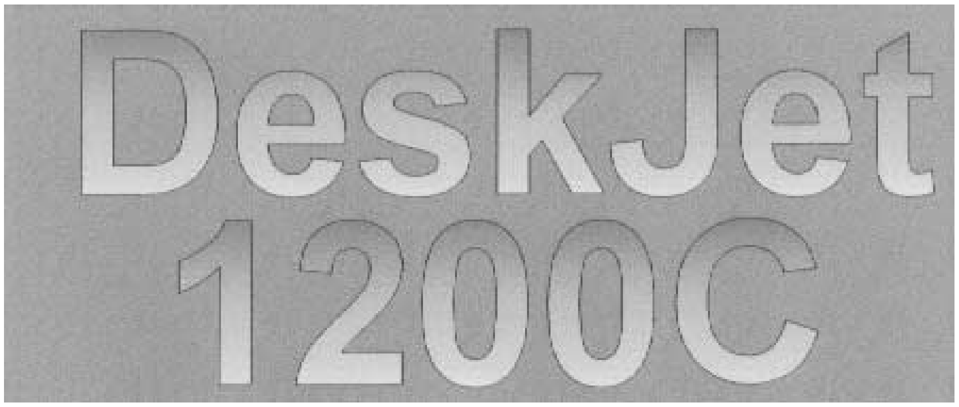


Fig. 9. The result of an operation similar to Fig. 8 involving gradient-textured text.

by not rendering the last row and last column of every pixel in a polygon defined under that model. For example, the grid-centered model of a rectangle from (1,1) to (3,4) is rendered as shown in Fig. 11.

The DeskJet 1200C printer also makes use of a high-precision polygon decomposition algorithm to break up complex user-defined polygons into simpler trapezoids and rectangles for the image processor to render using ROPs. The need for a high-precision polygon decomposition is mandated by the same set of problems that required an alternative pixel placement model. The polygon decomposition algorithm ensures that every polygon pixel is defined once and only once by maintaining integral and fractional pixel coordinate information to the desired level of precision.

Fig. 12 shows four adjacent rectangular patterns rendered using the ROP (Destination XOR Texture) under both of the pixel placement models. The overlapping pixels create extra black lines along the adjacent borders of the rectangles in the grid-intersection model. The problem is solved in the grid-centered model.

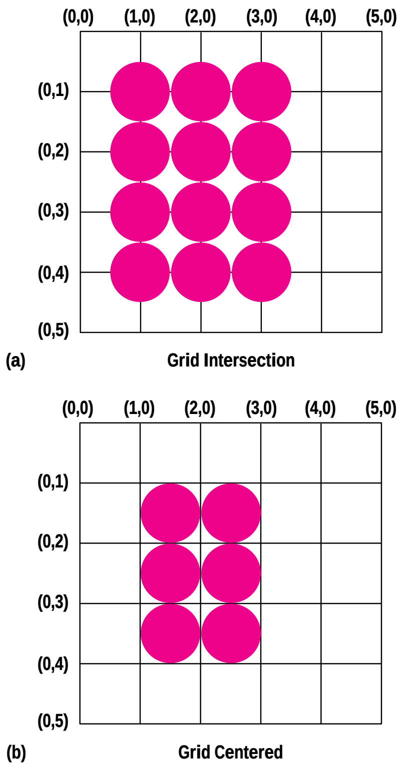


Fig. 10. Pixel placement methods. (a) Grid-intersection. (b) Grid-centered.

Predictable Rendering

When using ROPs to clip to an arbitrary path, it is not only necessary to be precise in physical pixel placement, but it is also necessary to render the objects in a predictable manner. In Fig. 13, a raster image has been rendered using a technique called *error diffusion*. Error diffusion takes an input raster image (each pixel can be defined as a 24-bit RGB color) and produces the output raster data in terms of dots on the physical printer destination. The output raster data is generated by looking at each pixel color and then propagating to the surrounding pixels the difference between the original color and the rendered physical color. In some implementations of error diffusion, a random factor is added to avoid any repeatable patterns in the output raster image. In these implementations, the same input raster image can produce a different output raster image every time it is rendered.

In Fig. 13, the raster image is being clipped into an oval shape in the same manner that the gradient shade was clipped in Fig. 8. The raster image is rendered and placed on the destination with ROP 90, then a black oval is placed on top using ROP 204, and finally, the image is rendered again onto the destination using ROP 90 (the outline of the oval is sent down separately from this clipping process). Fig. 13a shows what happens using error diffusion with a random factor. Rendering the raster image does not necessarily produce exactly the same raster output each time. Outside the black oval area, the image is not completely erased and will produce unpredictable results that appear as a ghost image of the original full raster image. Fig. 13b shows the desired

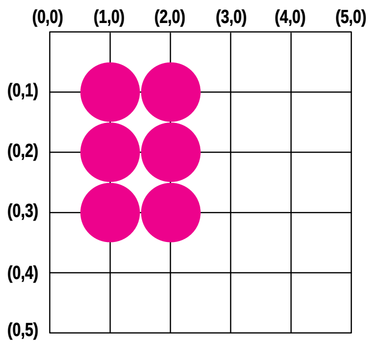


Fig. 11. Grid-centered model of a rectangle from (1,1) to (3,4).

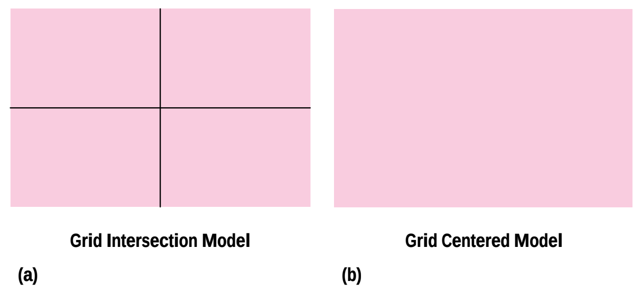


Fig. 12. (a) Rendering four adjacent rectangle patterns using the grid-intersection method results in extra black lines. (b) The problem is solved using the grid-centered method.

result when the output image is rendered in a way that is predictable and the image is completely erased outside of the oval.

Language Interface

For the sake of modularity and reuse, the HP DeskJet 1200C printer firmware is subdivided into major subsystems. Printer system "glue logic" firmware holds these subsystems together. The major subsystems include:

- Front-panel subsystem
- I/O subsystem
- Print engine subsystem
- Nonvolatile memory subsystem
- Printer Job Language (PJL) subsystem
- PCL and PostScript language subsystems.

Each subsystem is characterized by its own well-defined interface. The printer system is an amorphous entity with which each subsystem interacts. Thus, the printer system is

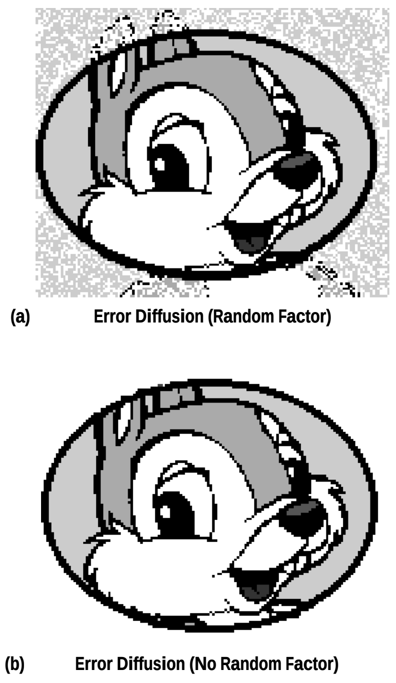

(a)
Error Diffusion (Random Factor)
(b)
Error Diffusion (No Random Factor)

Fig. 13. (a) Some implementations of error diffusion add a random factor, which can result in unpredictable ghost images when raster operations are performed. (b) Omitting the random factor results in predictable imaging.

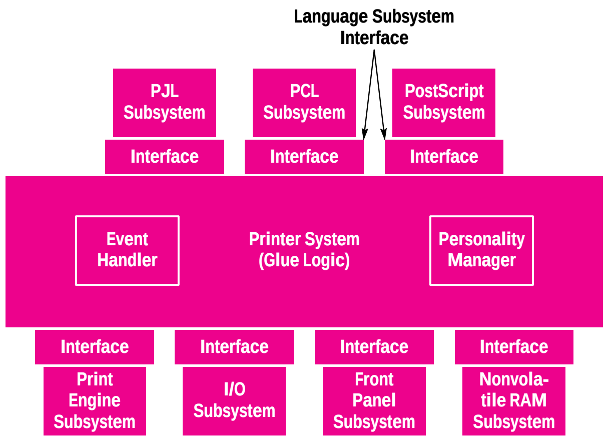


Fig. 14. HP DeskJet 1200C printer firmware architecture.

the mortar that holds the various subsystems together. It is thick in some places and thin in others, depending on how closely the subsystem interfaces line up. Two other blocks of firmware—the personality manager and the event handler—are not subsystems in a strict sense, since they lack their own well-defined interfaces. Instead, they use the interfaces of other subsystems. However, they are significant enough areas of the printer system that they are often referred to separately. Fig. 14 depicts the DeskJet 1200C printer firmware architecture.

The key subsystem in this architecture, in terms of connectivity, is the language subsystem. The DeskJet 1200C PS printer has two language subsystems: PCL 5C and PostScript Level 2. The language subsystem is the most significant because it indirectly interacts with all of the other subsystems, as well as the event handler and the personality manager. The interface between the language subsystem and the printer system, which we will call the *language interface*, is the subject of this section of this article.

At the most basic level, a language subsystem is an organized piece of software that accepts input language data from the printer system, interprets it, and produces a bitmap representation of an image, which is passed to the printer system to be transferred to the output media. In this respect, the language subsystem deals with I/O access for language parsing and with the print engine for imaging. The language subsystem is also involved in other interactions with the printer system such as personality management, access to nonvolatile memory, printer reconfiguration, printer system events and status, and language subsystem events and status.

Both PCL 5C and PostScript fit this definition of a language subsystem. The language interface was therefore designed to accommodate both. This has tremendous advantages. The overhead of having to maintain two different interfaces is eliminated. The printer system code in nearly all cases behaves identically regardless of whether the active language is PCL 5C or PostScript. This reduces the complexity of the printer system code. There is also only one language subsystem interface specification to maintain. Balancing the needs of PCL 5C and PostScript has been easy. There are only a few spots in the language interface that are specific to only one language.

Service Call	Description	When Called (Typically)
lang_init_req	Requests initialization of the language subsystem	System (re)initialization
enter_lang	Requests that the language become the interpreter of the input data stream.	Printer system determines the input stream belongs to the language.
lang_die	Requests language finish all imaging, free all resources, delete all processes, and otherwise prepare to die.	Printer system determines the input stream belongs to a different language.
media_fed	Acknowledges feed_media call. Indicates media is fed.	Printer system completes media loading.
swaths_imaged	Acknowledges swaths on image call have been imaged.	Printer system completes transfer of image to media.
async_periph_status	Notifies language of a change in peripheral status such as online, jam, etc.	Printer system has a change in status.

Fig. 15. Interface library routines provided by the language subsystem.

At the code level, the language interface consists of an information window and an interface library. The information window is a shared global data structure containing mostly static information about each side of the interface. The interface library contains two sets of routines. One set is provided by the language subsystem for the printer system to call. The other set is provided by the printer system for the language subsystem to call.

The information window and the interface library alone do not make the language interface complete. The language interface specification also includes a model-of-use section, which describes how the information window and interface library routines are used throughout various phases of print job execution. There is also a detailed section on media handling and imaging.

The information window for the DeskJet 1200C printer contains such items as: printer status, default image settings (paper size, character set, etc.), supported sources (input trays), destinations (output trays), and media sizes, as well as the printer's network name and type. The interface library contains the routines that handle the language subsystem's interactions with personality management, I/O access, imaging, status and events, and other interactions. Figs. 15 and 16 depict the two sets of routines that make up the interface library.

Model-of-Use Description

To best make use of the resources available, the primary one being RAM available to the language, the DeskJet 1200C printer system only has one language subsystem initialized at any given time. After power-on, the information window is initialized, and the printer goes online for the first time with no language subsystem initialized. When a print job enters the printer, the personality manager, with help from the printer job language subsystem, determines which

Service Call	Description	When Called (Typically)
lang_init_rsp	Acknowledges lang_init_req call. Ready to interpret data.	Language subsystem completes initialization.
get_input	Requests an input data buffer	Language parsing input data.
return_io_buf	Returns an input data buffer to printer system.	Language has "emptied" input data buffer.
return_unused_input	Returns a partially full input data buffer to printer system	Language does not want input data, possibly because of job boundary detection.
exit_lang	Indicates this language is no longer the interpreter for input data.	Language detects PJL universal exit language, or job_boundary flag set on get_input.
feed_media	Requests media be fed.	Imaging is to begin for a new page.
media_size	Requests dimensions of a media size.	Language wants to know initial imageable area of media.
image	Sends image swaths to the printer system to be printed.	Language has rendered, and is ready to print swaths.
send_output	Requests data be sent to the host system	Language wants to send data to host.
read_nvram	Reads data from nonvolatile RAM.	Language wants to retrieve data it has stored in nonvolatile RAM.
write_nvram	Writes data to nonvolatile RAM.	Language wants to store data into nonvolatile RAM.
printer_control	Flexible way to perform printer operations such as I/O reconfiguration and special media control.	Language wants to reset I/O configuration parameters or perform another special printer operation.
async_lang_status	Notifies printer system of a change in language status.	Language subsystem has a change in status.
restart	Requests restart of the language subsystem.	Language command to restart or catastrophic error occurred.

Fig. 16. Interface library routines provided by the printer system firmware.

language, PCL 5C or PostScript, the job needs to be sent to. Assuming there is no language initialized, the lang_init_req routine, provided by the target language subsystem, is called at a predetermined address.

The lang_init_req routine passes the resources the language subsystem will need to operate. The parameters passed to the language on the lang_init_req call specify the location of the information window, the RAM available to the language, pointers to operating system routines, a range for process priorities, and pointers to the printer system interface library routines, including the initialization acknowledgment call, lang_init_rsp. The language subsystem can call the read_nvram and write_nvram routines to retrieve or set information in the nonvolatile memory area allocated to the particular language. A call to the lang_init_rsp routine, provided by the printer system, indicates the language has finished initializing and is ready to accept a print job. The lang_init_rsp routine also passes the addresses of the remaining language subsystem interface library routines. Thus, with the exception of the well-known lang_init_req routine, the printer system and language subsystem are linked dynamically at run time.

When a print job is ready to be delivered and the appropriate language is initialized, the enter_lang routine is called by the printer system to indicate that the language subsystem is now the parser of the input data stream. The language subsystem uses the get_input routine to retrieve input data and the return_io_buf and return_unused_input routines to return I/O buffers back to the printer system. The language subsystem can become involved in imaging and in other interactions described below while parsing input data. The language subsystem remains the parser of the input data stream until it recognizes a valid exit event. The exit events on the DeskJet 1200C include the printer job language universal exit language sequence, a nonzero job boundary flag as a get_input routine parameter, an end-of-job character (control-D) for PostScript, and others. Once an exit event is recognized, the language subsystem calls the exit_lang routine to indicate that it no longer wishes to be the interpreter of the input data stream. Status reporting and any imaging that the language is busy doing continue following an exit_lang call.

After an exit_lang call, the personality manager examines the input data stream to determine where the next job goes. If the job is targeted for a language that is already initialized, the enter_lang routine is simply called again. This enables the language subsystem to begin parsing the next job while the previous job may still be printing. If the next job is for a different language than the one initialized, the lang_die routine for the initialized language is called. This instructs the language to finish any imaging it may be doing, return all I/O buffers back to the system, delete the processes it spawned, and otherwise prepare to die. Once a language returns from lang_die, the printer system is free to initialize a new language subsystem as described above.

In addition to the initialization, personality management, and language parsing side of the interface, the language subsystem is also involved in imaging. The language calls the feed_media routine to cause a sheet of media to be fed in preparation for the image to be placed on it. The request is queued by the printer system if it is busy printing a previous page. When the media is actually fed, the printer system calls the acknowledgment routine, media_fed. After feeding the media, the language subsystem renders the image into image buffers that cover a portion of the output page, and calls the image routine to pass the image data to the printer system for printing. The language subsystem need not wait for the media_fed routine before making image calls if it wants to work ahead. The swaths_imaged call is the printer system's way of acknowledging it has finished processing the image data and the language subsystem is now free to reuse the associated image buffers. An end-of-page parameter on the image routine indicates the printer system should eject the page following the placement of the image data on the media.

When the language is initialized, the printer system and the language subsystem exchange status and event signals through the async_periph_status and async_lang_status routines. The DeskJet 1200C printer system passes online, door open, jam, and other status, as well as the eject page and manual feed timeout events, to the language subsystem using async_periph_status. The DeskJet 1200C language subsystem passes its operating state (idle, busy, waiting) and a job end event using async_lang_status. The printer_control routine is used in the DeskJet 1200C to change the name and type of the printer as it appears on AppleTalk networks.

Documents

The language interface specification is broken into two documents: a device independent language interface specification and a device-specific document. This is to help facilitate reuse of the language interface. The device independent document provides a generic description of the information window, descriptions of the routines in the interface library, and the detailed model-of-use section, all of which apply to any device using the language interface. The device-specific document adds the level of detail required to specify the use of the language interface on a particular device such as the DeskJet 1200C printer.

For example, the feed_media routine, provided by the printer system to cause a sheet of media to be staged for printing, has parameters for specifying the source tray, destination tray, media size, media type, and print quality. The device independent language interface specification describes these parameters in a generic sense, but does not enumerate the valid values for a particular device. The DeskJet 1200C-specific document is the source of information as to what sources, destinations, media sizes, media types, and print quality values are valid for the DeskJet 1200C.

This approach to splitting the language interface specification has been beneficial. The DeskJet 1200C printer is the second of three devices to use this language interface so far. The HP PaintJet XL300 printer was the first, and the recently introduced HP DesignJet 650C large-format plotter also uses the language interface with the PostScript language (the DesignJet 650C does not support PCL 5C). The DeskJet 1200C and DesignJet 650C are able to reuse the device independent language interface developed for the PaintJet XL300 with only three minor enhancements. The DeskJet 1200C device-specific document was highly leveraged from the PaintJet XL300's with changes that reflect differences in the addresses for the lang_init_req routine, in the supported media sizes, in the front panels, and in other minor areas. The language interface is currently being considered for use in other products under development at various HP divisions.

Since the language interface supports two different language developers, it has to be simple, effective, efficient, and flexible. The simple view of a language subsystem that is the language interface fits the model of both PCL 5C and PostScript well. It is simple, and yet it is full-featured, supporting the best that PCL and PostScript have to offer. Because the language interface was designed with the complete firmware architecture in mind, it is efficient in both I/O access and imaging access. The split between a device independent document and a device-specific document strikes a balance between facilitating modularity and reuse and providing a realistic, flexible framework for implementation on a device.

Ever since version 1.0 of the language interface specification was released in May 1991, midway through the development of the PaintJet XL300 printer, modifications to the language interface have been through a formal change process. The change process provides a way to develop consensus and enhance the language interface in a controlled manner. Working with the PCL 5C development team and the PostScript team at Adobe Systems, the HP language interface team refined the interface using the change process. As each issue presented itself, the engineers involved would address the issue with a language interface change solution. A preliminary consensus would be reached, and a formal change would be submitted on an electronic bulletin board. After a review time, during which the change could undergo further review and revision, the change would be closed and incorporated into the implementation. The language interface specification, and perhaps more important, the language interface implementation, required constant communication with both groups of language developers. The DeskJet 1200C development team used the change process for both device independent document changes and device-specific document changes. The change process remains in place today to accommodate the needs of new products in development.

Screen Calibrator

With the advent of color output devices, many users have been disappointed with the consistency of the colors that their peripherals produce. Their carefully crafted screen image or scanned photograph is unacceptably distorted by many printers.

But how does one go about giving users the “no surprise” color that they need? The HP DeskJet 1200C printer achieves this by allowing the user to help characterize the particular CRT and display card. Armed with this knowledge, the DeskJet 1200C printer driver for Microsoft Windows can better match a user’s output with what appeared on the screen.

Consider two RGB triplets: 128,0,0 and 180,0,0. Other than saying that the latter is “redder” than the former to some unknown degree, those numbers alone convey very little information. When these RGB values are passed to various printers on the market today, one quickly discovers that their meaning is quite arbitrary; the actual colors output by different printers are surprisingly different.

The RGB triplets used by Microsoft Windows are not rigidly defined. Standards exist for specifying particular colors (these standards are often referred to as color spaces), but RGB values in most operating systems today are not specified in a standard color space.

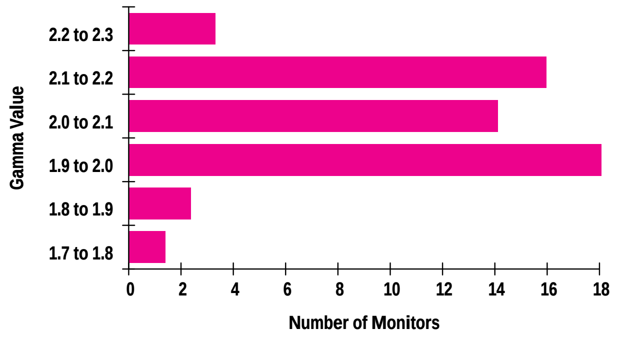


Fig. 17. Gamma values of identical monitors were found to vary considerably.

Returning to the problem at hand, the situation is even worse than it seems at first glance. Not only does every printer interpret RGB values differently, but so does every display card and every monitor. To top it all off, significant variations can be seen even among samples from the same brand and model of display equipment.

To get some idea of how significant these differences were, consider the case of several HP D1182 monitors that were tested in-house. Driven by a variety of different display cards, the monitors had gamma values (which will be explained later) that ranged from 1.7 to 2.3 (see Fig. 17). In general, a gamma difference of 0.1 was found to be noticeable. Overall variations (various monitors and display cards) were even more significant.

From this information, it was decided that no output matching technique could simply ignore these variations and still produce accurate hard-copy reproductions of CRT displayed data. Fortunately, HP Laboratories had already done some pioneering work in the field of CRT characterization.

If one can characterize a display system sufficiently, then it is possible to predict (in a standard color space) what the RGB triplet 128,0,0 actually means. By characterizing a color printer such that printer RGB triplets can be obtained to reproduce a standard color space value, one can then, in theory, accurately reproduce the color the user sees on the CRT on the output device.

The HP DeskJet 1200C, like all HP inkjet printers, shows relatively little color variation from one sample to the next. It can therefore be characterized statically with virtually no impact on the matching solution. There is, however, another problem that must be dealt with.

As one might imagine, different output technologies have different ranges of reproducible color. By way of illustration, if the RGB triplet 0,0,255 is specified on the screen, it will most likely produce a blue that the printer is unable to match. Conversely, when a DeskJet 1200C printer is given the triplet 255,255,0, it produces a yellow color that few CRTs can match.

The total collection of colors that a device can produce is referred to as the device’s gamut. Gamut mismatch is the problem described in the previous paragraph. Since we are using the display system as a reference point for the printer, the colors that the printer can produce but the CRT cannot produce are of no real concern.

The other part of the mismatch, CRT colors that the printer cannot accurately reproduce, is handled in the printer characterization. Since the printer characterization is done in a standard color space, information about nonreproducible colors can be imbedded in that characterization. Out-of-gamut colors are translated to colors the printer can produce in a process known as gamut scaling.

All decisions about the printer are made before the color matching software is ever put on the distribution disk. Characterizing a particular CRT, however, is another matter entirely. To match a CRT accurately, the DeskJet 1200C driver uses an HP-developed *screen calibrator tool* to obtain the information it needs about the display system of the user.

The introductory screen of the screen calibrator gives the user three options. The Standard Screen option gives default values for all the measurements the screen calibrator makes (loosely based on a Sony Trinitron display). Two calibration options—*precise* and *most precise*—give the user either a set of color screens or two black and white screens. The choice between the calibration options is a simple trade-off between ease of use and accuracy.

To better explain the measurements the screen calibrator makes, it is helpful to review how a CRT works. A CRT makes images by hitting phosphors with electron beams. In a color CRT, three different colors of phosphors (red, green, and blue) are struck by three electron beams. As more energy is applied to a patch, it glows brighter. From a distance, the colors of the three patches blend together to produce the varying hues the CRT can display.

Now imagine sending a series of colors to the CRT, beginning with 0,0,0 and running to 255,0,0. This series of reds will not, on virtually all display systems, produce 256 patches with equal-intensity steps between them. Furthermore, some values close to black, such as 5,0,0, often cause no noticeable activation of the phosphors. This threshold of activation, or deadband area, has been dubbed the *dc offset* of the display system.

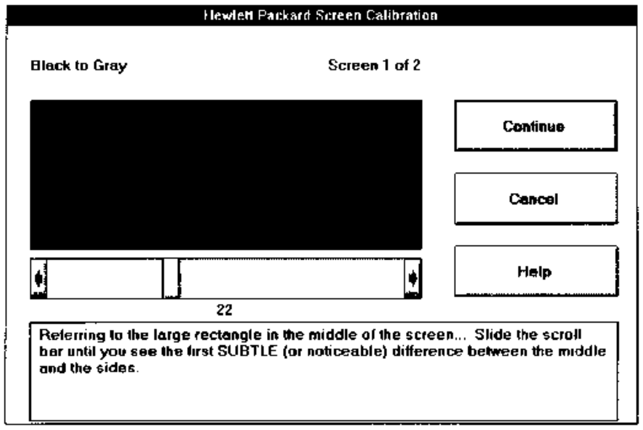
The color intensities produced after passing the activation threshold follow an exponential curve:

$$I_A = I_I^{\gamma}.$$

where I_A is the actual output intensity and I_I is the intensity level desired. The intensities are normalized and have values between 0 and 1. The exponent of this curve, known as gamma, is the important piece of data to derive. Using the screen calibrator, users perform tests that determine the dc offset and gamma of a display system.

Figs. 18 and 19 show how the screen calibrator interacts with the user to determine the gamma and dc offset. In Fig. 18, the user moves the slider until the center of the rectangle can just be distinguished from the completely black sides. This allows the screen calibrator to measure the dc offset of the display system.

The gamma determination is considerably more challenging to the user. By alternating colored and black lines on the



Referring to the large rectangle in the middle of the screen... Slide the scroll bar until you see the first SUBTLE (or noticeable) difference between the middle and the sides.

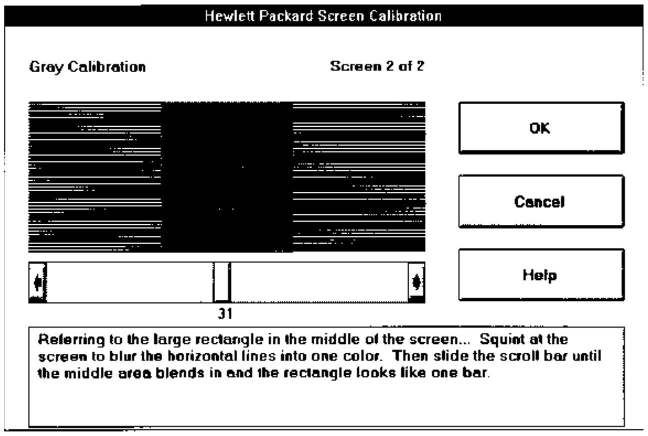
Fig. 18. The screen calibrator tool lets the user help match the DeskJet 1200C printed output to the monitor screen. This screen is for determination of the monitor's dc offset.

sides of the central patch, different intensities can be accurately created (see Fig. 19). The problem is that the user must somehow coerce the alternating lines into a solid patch of color. This can be accomplished by the sophisticated visual process of squinting, as suggested by the instructions, but moving away from the screen or removing one's eyeglasses can sometimes serve the same purpose.

Having successfully squinted and adjusted, the user has made the center patch match the wide patches and the whole rectangle appears to be of one intensity. This allows the screen calibrator to correlate the actual intensity—the bars—with the value that the display card and monitor need to display that intensity. A curve is then fit to the points to derive gamma.

The difference between the precise and most precise calibrations is twofold. First, the most precise calibration determines the gamma of the red, green, and blue channels of the system separately, while the precise calibration uses white, effectively using the user's eye to average the three channels.

The second difference between the calibration options allows adjustments for the kind of light that illuminates the printed output. The user's DeskJet 1200C prints a purple-colored



Referring to the large rectangle in the middle of the screen... Squint at the screen to blur the horizontal lines into one color. Then slide the scroll bar until the middle area blends in and the rectangle looks like one bar.

Fig. 19. Screen calibrator tool screen for determination of the gamma of the monitor.

patch, which the user then compares to a series of patches on the screen. Depending on the kind of lighting around the system, the user will see the printed patch as either more bluish or more purplish than the displayed patches. The user selects the displayed patch that most closely matches the printed patch.

Once all this interactive information is gathered, color maps are generated for each combination of dithering and media type. This is necessary because each of these combinations has a different effect on the colors produced by the printer.

Two main classes of color maps are generated. Match to Screen maps faithfully represent on paper what appears on the user's screen. More Vivid maps increase the color saturation of the output.

Thus, by interacting with users to determine the characteristics of their specific display systems, the screen calibrator tool of the DeskJet 1200C Windows driver helps deliver the accuracy and consistency that users demand.

Acknowledgments
The authors of this article would like to acknowledge all those who contributed to the DeskJet 1200C connectivity effort—both the PCL 5C language team and the driver development teams. They would also like to thank those who contributed to the completion and support of this article: Katherine Albitz, Lee Jackson, Tom Smith, John Stout, and Jerry Villone. Frank Bockman and Craig Blackwood of the desktop color team contributed to the design and implementation of the screen calibrator. The color management development team would like to thank Ricardo Motta and Andrew Fitzhugh of HP Laboratories for their technological help, and Al Shanmugam and Daniel Kuehler for facilitating the screen calibrator's development and use. Also of note was Steve Breidenbach's tireless effort to turn a completely obtuse user interface into something much more friendly.

Reference

1. *Tile Vector to Raster Conversion Method,* U.S. Patent #5,028,848, July 2, 1991.