| (54) Title | Printing control method and apparatus |
| (51) International Patent Classification(s) | G06F 3/12 (2006.01) |
| (21) Application No: | 2008229690 | (22) Date of Filing: | 2008.09.29 |
| (43) Publication Date: | 2010.04.15 |
| (43) Publication Journal Date: | 2010.04.15 |
| (71) Applicant(s) | Canon Kabushiki Kaisha |
| (72) Inventor(s) | Wu, Sheng; Smith, David Christopher |
| (74) Agent / Attorney | Spruson & Ferguson, Level 35 St Martins Tower 31 Market Street, Sydney, NSW, 2000 |
ABSTRACT

PRINTING CONTROL METHOD AND APPARATUS

Disclosed is a method of rendering a page for printing. The method determines (200) a color processing mode for each band on the page based on the type and color values of the graphical contents of that band, wherein at least one band is determined to have an unknown color processing mode. A band with an unknown color processing mode is rendered on the page firstly in a 1 bit-per-pixel mode to output pixel data comprising only two color values (eg. black and/or white). The method detects (560) during the rendering that at least a third color needs to be output. The method expands (565) the already output 1 bit-per-pixel data of the band into contone format and renders the remainder of the pixels of the band in contone mode.
Figure 1
Name and Address of Applicant: Canon Kabushiki Kaisha, of 30-2, Shimomaruko 3-chome, Ohta-ku, Tokyo, 146, Japan

Actual Inventor(s): David Christopher Smith, Sheng Wu

Address for Service: Spruson & Ferguson
St Martins Tower Level 35
31 Market Street
Sydney NSW 2000
(CCN 3710000177)

Invention Title: Printing control method and apparatus

The following statement is a full description of this invention, including the best method of performing it known to me/us:
PRINTING CONTROL METHOD AND APPARATUS

TECHNICAL FIELD OF INVENTION

The present invention relates to graphic rendering, and in particular, relates to render different sections in different color processing mode.

RELATED BACKGROUND ART

The speed and cost of printing a monochrome page is much faster and cheaper than a color page. There are conventional methods to determine monochrome sections in a color page before printing so to print different sections in different color processing mode.

SUMMARY OF THE INVENTION

In accordance with one aspect of the present invention there is provided a method of rendering a page for printing said method comprising the steps of:

- determining a color processing mode for each band on said page based on the type and color values of the graphical contents of that band, wherein at least one band is determined to have an unknown color processing mode;
- rendering said band with unknown color processing mode on said page firstly in a 1 bit-per-pixel mode to output pixel data comprising only two color values;
- detecting during said rendering that a third color needs to be output;
- expanding the already output 1 bit-per-pixel data of said band into contone format; and
- rendering the rest pixels of said band in contone mode.

Other aspects are also disclosed.
BRIEF DESCRIPTION OF THE DRAWINGS

One or more embodiments of the invention will now be described with reference to the following drawings, in which:

Fig. 1 is a schematic flow diagram illustrating the components and control flow involved in rendering a PDL (Page Description Language) document according to one embodiment of the invention;

Fig. 2 is a schematic flow diagram illustrating a method of building a display list for a renderer to render, with a color processing mode determination procedure, as used in the method of Fig. 1;

Fig. 3 is a schematic flow diagram illustrating the control flow involved in rendering a display list using different color processing mode specified in the display list, as used in the method of Fig. 1;

Fig. 4 is a schematic flow diagram illustrating a method of rendering a band from a display list in a known contone or 1 bit-per-pixel color space, as used in the method of Fig. 3;

Fig. 5 is a schematic flow diagram illustrating a method of rendering a band from a display list in an unknown color space in an adaptive way, as used in the method of Fig. 3;

Fig. 6 is a schematic flow diagram illustrating the components and control flow involved in packing and sending the pixel data generated by the process of Fig. 3, as used in the method of Fig. 1;

Fig. 7 shows an example page which contains several objects to be printed; and

Fig. 8 is a schematic block diagram of a general purpose computer on which the embodiments of the invention may be practised.
DETAILED DESCRIPTION OF EMBODIMENTS OF THE INVENTION

Figure 8 illustrates the components of a general purpose computer in one of the embodiments of this invention on which a process 100 depicted in Fig. 1 may be executed. The computer 100 in this embodiment has a computer module 801, input devices including a keyboard 802 and a computer mouse 803 connected to the Input/Output Interface 813, output devices including a display monitor 814 and a pair of speakers 817 connected to the Audio-Video Interface 807, and a printer 816 connected to the Input/Output Interface 808. Inside the computer module 801, the centre processor 805 is connected to the I/O Interfaces 808, 813 and Audio-Video Interface 807 by the centre bus 804. The internal Memory unit 806, the CD-ROM driver 812 and the external storage unit 809 which contains the Hard Disk Drive 810, Floppy Disk Drive 811, are also connected to the Processor 805 through the main bus 804.

Figure 1 illustrates the components and general control flow in one of the embodiment of this invention generally present in a system required to render a PDL document 105. A PDL interpreter 110 accepts PDL document 105, interprets its contents, and sends the graphical objects to be printed in a format of drawing commands 115 to a display list builder 120. The display list builder analyses the graphical objects and creates a display list 125, which consists mostly of drawing instructions, as well as color processing mode information, in a format suitable for the renderer 130 to process. The renderer 130 then accepts display list 125, renders the display list 125, and produces pixel data of each band 135 into buffers. The buffers 135 are then processed and packed by the output engine 140, into page image packets 145. The packets are sent to the printer 150 via links between the printing server and the printer. The printer 150 processes the packets 145 and prints the physical pixels on a sheet of paper.
The PDL interpreter 110, display list builder 120, renderer 130 and output engine 140 can be, and are not limited to be implemented in the printer driver software running on a general purpose computer, or in printer hardware modules. In this embodiment which is described in this description, they are software modules in the printer driver running on the general purpose computer 800. The processor 805 and the memory 806 are the resources in the general purpose computer 800 which are used to execute the printer driver software.

Figure 2 illustrates a color processing mode determination method 200, as used in the display list builder 120 in process 100. The display list builder 120 accepts the drawing commands 115 from PDL interpreter 110. The drawing commands describe the graphical objects to be printed on a page. An example page, which is initially divided into 4 bands with each band consists 100 scanlines, is given in Figure 7. The corresponding drawing commands to describe the 4 objects on the page are given in Figure 7 too.

At step 210 the processor 805 of display list builder 120 initialises a band_color_mode[] array, in the computer memory 806, containing the color processing mode that will be used for each band, with default value of “none” meaning no objects on this band (none=0x00, 1bpp=0x01, contone=0x02, unknown=0x03 in this example). The display list builder 120 accepts the printing objects in the drawing commands 115 one at a time. For each object, the display list builder 120 converts the object color into render color space at step 220. The color conversion module examines the color attribute of the object during converting and reports back to the display list builder 120 when the conversion finishes. If the object only contains only two color values, such as
black and/or white colors, then the color mode of this object is reported as “1bpp”. Otherwise the color mode of this object is reported as “contone” (ie. continuous tone).

At step 230 the display list builder 120 analyses the range of the bands that this object will be printed onto, and updates corresponding member of the band_color_mode[] array with the current object’s color attribute in step 240. For the given example, the corresponding band_color_mode[cur_object_band_range] are OR-ed by the current object color mode. The bands only contains black/white objects are set to be “1bpp” mode. The bands only contains colorful objects are set to be “contone” mode. The bands that contain both black/white and colorful objects are set to be “unknown” mode. The proceeding described herein is performed on at least one band set to be “unknown”.

In step 250, the display list builder 120 process the object according to its type and add corresponding edges and fills into the result display list 125. The edges describe the boundary of the object, while the fills describe the data to “fill” the objects. The display list 125 maintains the edge list of all the edges, the fills list of all the fills, and level information which describes the rules of filling the data onto each area, i.e. priority of each objects and compositing operations etc. Some objects, like texts, can be represented in bit bitmaps directly. An example display list that describes the page in Figure 7 is given below. At step 260, the display list builder checks if there is more objects to be processed. After all objects are processed, the result display list 125 is sent to renderer 130 at step 270.

```
Display List 125:
edge_list:
25  
   
edge0:
1420211_1
```
{  
    start_point = (100, 30);  
    points_list = {(200, 30); (200, 60); (100, 60); (100, 30)};  
    level = level0;  
}

edge1:  
{
    start_point = (60, 120);  
    points_list = {(260, 120); (260, 230); (60, 230); (60, 120)};  
    level = level1;  
}

ddge2:  
{
    start_point = (100, 180);  
    points_list = {(200, 180); (200, 190); (100, 190); (100, 180)};  
    level = level2;  
}

ddge3:  
{
    start_point = (100, 205);  
    points_list = {(200, 205); (200, 330); (100, 330); (100, 205)};  
    level = level3;  
}

level_list:  
{
    level0:  
    {  
        priority = 0;  
        fill = fill0;  
    };  
    level1:  
    {  
        priority = 1;  
        fill = fill1;  
    };

25
30
35
level2:
{
    priority = 2;
    fill = fill2;
};
level3:
{
    priority = 3;
    fill = fill3;
};

fill_list:
{
    fill0:
    {
        type = image;
        color_space = gray;
        bit_depth = 1;
        image_value = 0x.....;
    }
    fill1:
    {
        type = flat_color;
        color_space = gray;
        bit_depth = 1;
        color_value = 0x0;
    }
    fill2:
    {
        type = image;
        color_space = RGB;
        bit_depth = 8;
        color_value = 0x....;
    }
Figure 3 illustrates the control flow of a render procedure 300 presented inside a renderer one embodiment of the invention. The renderer 130 receives the display 125 from the display list builder 120 at step 310. At step 320 the renderer 130 initializes variable cur_band to be 0. For every band to be rendered, the renderer 130 sets the current band color processing mode to be band_color_mode[cur_band] from the display list 125 at step 330, and also sets the num_scanline_finished which records the number of scanline successfully finished rendering under current color processing mode in this band. Then the renderer 130 renders the current band in the just set band_buffer_type mode, which are described in detail in Figure 4 and Figure 5. Figure 4 (procedure 400) illustrates the method of rendering a band in a fixed color processing mode, i.e. "1bpp" or "contone" mode. Figure 5 (procedure 500) illustrates the method of rendering a band, which has been set to be "unknown" in band_color_mode[] array, in an adaptive color
processing mode. After step 340 the pixel data of the band 135 is generated and sent to output engine 140. The renderer 130 then checks if the current band has finished rendering, which will be described in detail in Figure 5 and Figure 6. If no, the renderer 130 sets the band_buffer_type to be “contone” at step 335 and renders the un-finished band in “contone” mode back at step 340; If yes, the renderer 130 increases the cur_band as in step 350, and checks if there is more band needs to be rendered at step 360. After all bands are rendered into pixel data, the renderer finishes rendering the page at step 370.

Figure 4 illustrates the method 400 of rendering a band in a fixed color processing mode, i.e. either “1bpp” or “contone” mode, as used in this invention. At step 410 the renderer 130 receives parameters 405 from procedure 300, including the display list 125, a buffer to put rendered pixels in, and the rendering color mode etc. At step 420 the renderer 130 analyses the display list and makes an active edge list for the current scanline. The Active Edge List (AEL) contains all the edges which are currently active on a scanline, and is maintained in increasing X-order. Pixel-runs are generated as the spans of pixels between 2 edges in the AEL. A scanline can have multiple pixel-runs. At step 430, the renderer 130 generates the Active Level List (ALL) for this pixel-run, and does the level processing based on the ALL. At step 435, if the ALL only contains one level, or if the top-most level is opaque, then the contents on that level is directly output as the content of the span for step 450; Otherwise, the top level is not opaque, all the non-opaque levels until the most-top opaque one are added to a compositing list, which is sent to compositing procedure. At step 440 the renderer 130 composites the levels on the composite list and generates an output for this span. At 450, the result pixel values are filled into the band buffer 455, in the computer memory.
806, in the current band_buffer_type format. At step 460, the renderer 130 checks if there are more pixel-runs for this band to render. If so, the renderer 130 updates the num_scanline_finished if a full scanline is finished at step 465, and goes back to step 420 to generate the next pixel-run; If not, the renderer 130 outputs to the output engine the rendered pixel data 135 in the band buffer along with the size of the buffer, the color space type of the pixel value and the number of scanline rendered of this band at step 470.

Figure 5 illustrates the method 500 of rendering a band in an unknown color processing mode tagged by the display list builder 120. At step 510 the renderer 130 receives parameters 405 from procedure 300, including the display list 125, a buffer to put rendered pixels in, and the band_buffer_type, which is "unknown", and the num_scanline_finished, which is 0. At step 515 the renderer 130 sets the output buffer_type to be "1bpp". The renderer will try to render output in "1bpp" format. The parameter num_scanline_finished records the number of scanlines that has finished rendering during the rendering procedure. Same as in 420, at step 520 the renderer 130 analyses the display list and makes an Active Edge List (AEL) for the current scanline, and generates the next pixel-run based on the AEL. At step 530, the renderer 130 generates the Active Level List (ALL) for this pixel-run, and does the level processing based on the ALL. At step 535, if the ALL only contains one level, or if the top-most level is opaque, then the contents on that level is directly output as the content of the span for step 545. Otherwise, the top level is not opaque; all the non-opaque levels until the most-top opaque one are added to a compositing list, which is sent to compositing procedure. At step 540 the levels on the compositing list are composited and the result pixel values are generated for this pixel-run.
At step 545, the renderer checks the result pixel value from step 535 or 540 if they are non-black/white color before writing to buffer. If the result pixel value is black/white only, then it is saved and filled into the band buffer 570 in 1 bit-per-pixel format at step 550. The parameter num_scanline_finished is increased by 1 if a whole scanline is finished at step 555. Then the renderer checks if there are more pixel-runs need to be rendered at step 525. If there are more pixels, the control goes back to step 520. If not, then it means all pixels in this band are successfully rendered in 1 bit-per-pixel mode; the renderer then goes to step 575: outputs the fulfilled band buffer along with the buffer size and buffer_type and num_scanline_finished to the output engine, shown in procedure 600, Figure 6.

At step 545, if the result pixel value from step 535 or 540 contains colorful info, then the control enters step 560 as a “recovery procedure”. Firstly the num_scanline_finished is checked at step 560, to see how many scanlines are already successfully rendered in “1bpp” mode. There is a pre-set threshold parameter. If num_scanline_finished is larger than this threshold, then the band buffer 570 and the buffer size for the already rendered num_scanline_finished lines is output to output engine at step 575, along with the buffer_type as “1bpp”.

If the num_scanline_finished is detected as being less than or equal to the pre-set threshold at step 560, then the control goes to step 565, where the renderer expands the already generated 1 bit-per-pixel data in the band buffer 570 into contone format in band buffer 595 thus providing for an output of at least a third color for the band. The 1bpp band buffer 570 could use the same memory space as the contone band buffer 595. The renderer when initialising the band buffer for the “unknown” band in step 330 can simply prepare a buffer with the size for a band of contone data. At step 580, the
renderer 130 changes the buffer_type from "1bpp" to be "contone", and calls process 400 to finish the rest of the band in contone color processing mode, where the rest pixels of the band are rendered in contone value and output to output engine.

The threshold parameter at step 560 could be determined and set differently for each printing job, depending on the type of the documents, to improve efficiency.

Figure 6 illustrates the method 600 in an output engine 140 of packing and sending the pixel data of the bands generated by the process 300 as used by this invention. At step 610 the output engine 140 initialises the cur_band to 0. At step 620 the output engine 140 receives the pixel data of the band 135 from the renderer 130.

Then the buffer_type of the band is checked at step 625. If it’s a contone color data band, the pixel data of the band 135 is sent for post-render color conversion and dithering process at step 650, where the contone data is converted into output color space, normally CMYK, and is dithered for printing. At 660 the dithered data is packed to contone package 145 in the communication protocol between the printer and the printing server, indicating the printer of the pixel data of the band and the format of the data. The package 145 is sent to the printer via connections between the printing server and the printer at step 640.

If the buffer_type of the band is "1bpp" at step 625, then the pixel data of the band 135 is directly packed to 1bpp package 145 in the communication protocol between the printer and the printing server, without the need of color converting and dithering. The printer has the ability to process and print the 1 bit-per-pixel color data directly, and these pixels don’t need to be dithered. This saves the expensive time spending on the post-render color conversion and dithering. Also the 1bpp packages are much smaller than contone packages, thus faster to transfer. The package 145 is then sent to printer via
the connections between the printer server and the printer at step 640. The process on
the printer for the 1bpp bands is also easier.

At step 670 the output engine 140 checks if the band is just the first part of the
initially "unknown" band, by checking if the buffer_type is "1bpp" and the
num_scanline_finished less than the full_band_size. If it is the first part of the
"unknown" band then the control goes back to step 620 and receives the next part of the
band from the renderer 130. Otherwise the current band is finished. The output engine
140 increases the cur_band at step 680, and checks if there is more band needs to be
packed at step 685. If there is more band, the control goes back to step 620 to receive
the pixel data of the next band. Otherwise the printing job is finished at step 690.

**Example of the rendering process**

Below is the description of the rendering process of the page presented in
Figure 7, as an example to the rendering process described in 300, 400, 500 and 600
above. In this example only the vertical edges appear in the Active Edge List and are
numbered as e0 to e7. Levels and Fills are using the same names as in the display list
element in the previous section. And the pre-set threshold variable is set to 20.

Band 0 only has black texts and is tagged as "1bpp" in band_color_mode[0] in
display list. So at step 340, process 400 is called. For scanline 1 to 30, no level is linked
and the background color white is filled in 1 bit-per-pixel format.

For scanline 31, at step 420, the Active Edge List is decided as {e0, e1}. Three
pixel-runs are generated for this scanline: pixel 1 to 100, pixel 101 to 200, and pixel 201
to 282. For pixel-run 1 to 100, no level is linked so background color white is filled.
For pixel-run 101 to 200, at step 430, the Active Level List is decided as {level0}. Only
one opaque level "level0" is on the list, so at step 435 the text bitmaps in "fill0" which is
linked to "level0" is retrieved and directly output in 1 bit-per-pixel format to the band buffer at step 450. For pixel run 201 to 282, background color white is filled. This process repeats from scanline 32 to scanline 60.

For scanline 61 to 100, no level is linked and the background color white is filled. Then the rendering of band 0 finishes at step 470, and the control goes back to step 350, where the cur_band is increased by 1.

Band 1 contains both black/white only object (obj1) and color object (obj2), and is tagged as "unknown" in band_color_mode[1] in display list. So at step 340, process 500 is called. At step 515, the buffer_type of the band is set to "1bpp". For scanline 101 to 120, no level is linked and the background color white is filled in 1 bit-per-pixel format. _num_scanline_finished_ is increased by 1 for every filled scanline.

For scanline 121 to 180, at step 520, the Active Edge List is decided as {e2, e3}. Three pixel-runs are generated for this scanline: pixel 1 to 60, pixel 61 to 260, and pixel 261 to 282. For pixel-run 1 to 60, no level is linked so background color white is filled in 1 bit-per-pixel format at step 550. For pixel-run 61 to 260, at step 530, the Active Level List is decided as {level1}. Only one opaque level "level1" is on the list, so at step 535 the color value of "fill1" which is linked to "level1" is retrieved and sent to step 545. Because it is Black, so the control goes to step 550 where the black color is filled into the band buffer for this pixel run in 1 bit-per-pixel format. At step 555 the _num_scanline_finished_ is not increased because there is more pixel-run on this scanline. For the next pixel-run, from pixel 261 to 282, no level is linked so background color white is filled in 1 bit-per-pixel format at step 550. Then _num_scanline_finished_ is increased by 1 at step 555.
For scanline 181, at step 520, the Active Edge List is decided as \{e2, e4, e5, e3\}.

Five pixel-runs are generated for this scanline: pixel 1 to 60, pixel 61 to 100, pixel 101 to 200, pixel 201 to 260, and pixel 261 to 282. For pixel-run 1 to 60, no level is linked so background color white is filled in 1 bit-per-pixel format at step 550. For the next pixel-run, from 61 to 100, the process is the same as the process of the pixel-run 61 to 260 in scanline 121, as described in the previous paragraph. At step 530, the Active Level List is decided as \{level1\}. Only one opaque level “level1” is on the list, so at step 535 the color value of “fill1” which is linked to “level1” is retrieved and sent to step 545. Because it is Black, so the control goes to step 550 where the black color is filled into the band buffer for this pixel run in 1 bit-per-pixel format. At step 555 the num_scanline_finished is not increased because there is more pixel-run on this scanline.

For the next pixel-run, from pixel 101 to 200, the Active Level List is decided as \{level2\}. Only one opaque level is on the list so at step 535 the color value of “fill2” which is linked to “level2” is retrieved and sent to step 545. The color value retrieved is contone color value so the control goes to step 560. By now the num_scanline_finished is updated to 80 which is greater than the threshold of 20, so the control goes to step 575. The renderer 130 output the already generated band buffer of the scanline from 101 to 180 in the format of 1 bit-per-pixel back to the output engine.

The control returns to step 345 in procedure 300, where the returned buffer_type and num_scanline_finished is checked to determine if this is the first part of the “unknown” band. In this case the buffer_type is “1bpp” and um_scanline_finished is less than full_band_size, which means the band is not finished yet, so the program goes to step 335, where the buffer_type is changed to “contone”. The process 400 is being called at step 340 for the scanline 181 to 200.
For scanline 181, at step 420, the Active Edge List is decided as \{e2, e4, e5, e3\}. Five pixel-runs are generated for this scanline: pixel 1 to 60, pixel 61 to 100, pixel 101 to 200, pixel 201 to 260, and pixel 261 to 282. For pixel-run 1 to 60, no level is linked so background color white is filled in 1 contone format at step 450. For the next pixel-run 61 to 100 at step 430, the Active Level List is decided as \{level1\}. Only one opaque level “level1” is on the list, so at step 435 the color value of “fill1” which is linked to “level1” is retrieved and sent to step 450. Because now the buffer_type is contone, so the color value retrieved is filled into the band buffer 455 in contone format. At step 465 the num_scanline_finished is not increased because there is more pixel-run on this scanline. For the next pixel-run, from pixel 101 to 200, the Active Level List is decided as \{level2\}. Only one opaque level is on the list so at step 535 the color value of “fill2” which is linked to “level2” is retrieved and filled in the band buffer 455 in contone format. For the next pixel-run 201 to 260 at step 430, the Active Level List is decided as \{level1\}. Only one opaque level “level1” is on the list, so at step 435 the color value of “fill1” which is linked to “level1” is retrieved and sent to step 450. Because now the buffer_type is contone, so the color value retrieved is filled into the band buffer 455 in contone format. For pixel-run 261 to 282, no level is linked so background color white is filled in 1 contone format at step 450. The num_scanline_finished is then increased by 1 at step 465.

The process for scanline 182 to 190 is the same as the process for scanline 181. The process for scanline 191 to 200 is the similar as the process for scanline 181 except for there was 3 pixel-runs instead of 5. The contone band buffer of this part of the band is output to output engine at step 470.
Band 2 contains both black/white only object (obj1) and color object (obj3), and is tagged as "unknown" in band_color_mode[2] in display list. So at step 340, process 500 is called. At step 515, the buffer_type of the band is set to "1bpp". For scanline 201 to 204, the same process as scanline 121 to 180 is performed. 1 bit-per-pixel format data is filled in the band buffer 570 for scanline 201 to 204.

For scanline 205, at step 520, the Active Edge List is decided as \{e2, e6, e7, e3\}. Five pixel-runs are generated for this scanline: pixel 1 to 60, pixel 61 to 100, pixel 101 to 200, pixel 201 to 260, and pixel 261 to 282. For pixel-run 1 to 60, no level is linked so background color white is filled in 1 bit-per-pixel format at step 550. For the next pixel-run, from 61 to 100, the process is the same as the process of the pixel-run 61 to 260 in scanline 121, as described in the previous paragraph. At step 530, the Active Level List is decided as \{level1\}. Only one opaque level "level1" is on the list, so at step 535 the color value of "fill1" which is linked to "level1" is retrieved and sent to step 545. Because it is Black, so the control goes to step 550 where the black color is filled into the band buffer for this pixel run in 1 bit-per-pixel format. At step 555 the num_scanline_finished is not increased because there is more pixel-run on this scanline. For the next pixel-run, from pixel 101 to 200, the Active Level List is decided as \{level3\}. Only one opaque level is on the list so at step 535 the color value of "fill3" which is linked to "level3" is retrieved and sent to step 545. The color value retrieved is contone color value so the control goes to step 560. By now the num_scanline_finished is updated to 4 which is less than the threshold of 20, so the control goes to step 565. The renderer 130 expands the pixel data from the band buffer 570 from 1 bit-per-pixel to contone format, and saves into band buffer 595. As described in previous section, the 1bpp band buffer 570 and contone band buffer 595 can share the same memory space.
At step 580 the renderer set the buffer_type to "contone" and performs the procedure 400, where the renderer 130 continues rendering the scanlines from 205 to 300 in "contone" mode and outputs the contone band buffer back to the output engine at step 470.

Band 3 only has color image and is tagged as "contone" in band_color_mode[3] in display list. So at step 340, process 400 is called. The scanlines from 301 to 330 are rendered the same as scanlines from 210 to 300. For scanline 1 to 30, no level is linked and the background color white is filled in contone format. The contone band buffer of this band is output to output engine at step 470. Then the control goes back to step 360 in process 300, where the renderer 130 determined that all bands have finished rendering and goes to step 370.

The forgoing describes only some embodiments of the present invention and modifications may be made thereto without departing from the scope of the present disclosure.
THE CLAIMS DEFINING THE INVENTION ARE AS FOLLOWS:

1. A method of rendering a page for printing said method comprising the steps of:
   determining a color processing mode for each band on said page based on the type and color values of the graphical contents of that band, wherein at least one band is determined to have an unknown color processing mode;
   rendering said band with unknown color processing mode on said page firstly in a 1 bit-per-pixel mode to output pixel data comprising only two color values;
   detecting during said rendering that a third color needs to be output;
   expanding the already output 1 bit-per-pixel data of said band into contone format; and
   rendering the rest pixels of said band in contone mode.

2. A method of rendering a page for printing substantially as described herein with reference to the drawings.

3. Computerised apparatus adapted to perform the method of claim 1 or 2.

4. A computer readable storage medium having a program recorded thereon, the program being executable by computerised apparatus to perform a method according to claim 1 or 2.

Dated this 29th day of September 2008

CANON KABUSHIKI KAISHA

Patent Attorneys for the Applicant

Spruron & Ferguson
Figure 2
Figure 3
Figure 4
Figure 5
Figure 6
Figure 7

Drawing Commands:

Obj0:
{
  bounding_box = \{(100, 30), (200, 60)\};
  obj_type = text;
  color_mode = 1bpp;
  color_value = 0x01;
  glyph_bitmap = "0x.......";
}

Obj1:
{
  bounding_box = \{(60, 120), (260, 230)\};
  obj_type = graphics;
  color_mode = 1bpp;
  color_value = 0x01;
  lines = {...};
}

Obj2:
{
  bounding_box = \{(100, 180), (200, 190)\};
  obj_type = image;
  color_mode = RGB;
  image_value = "0x.......";
}

Obj3:
{
  bounding_box = \{(100, 205), (200, 330)\};
  obj_type = image;
  color_mode = RGB;
  image_value = "0x.......";
}

**e_0 to e_7:** vertical edges that appear in the display list.