(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) **EP 1 772 850 A1**

(12) **EUROPEAN PATENT APPLICATION**

(43) Date of publication:
11.04.2007 Bulletin 2007/15

(51) Int Cl.:
G09G 5/00 *(2006.01)*

(21) Application number: **05256266.7**

(22) Date of filing: **07.10.2005**

(84) Designated Contracting States:
AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HU IE IS IT LI LT LU LV MC NL PL PT RO SE SI SK TR
Designated Extension States:
AL BA HR MK YU

(71) Applicant: **BRITISH TELECOMMUNICATIONS public limited company**
London EC1N 7AJ (GB)

(72) Inventor: **The designation of the inventor has not yet been filed**

(74) Representative: **Cardus, Alan Peter et al**
BT Group Legal
Intellectual Property Department, BT Centre
PP C5A
81 Newgate Street
London EC1A 7AJ (GB)

(54) Burn in reduction in a display device

(57) A signal processor for reducing wear in display devices and, in particular, to reduce burn-in. The signal processor providing a signal to a display device defining the image to be displayed. The signal processor processing the signal to cause temporary reductions in the intensity of pixels in the displayed image. The image is divided into a plurality of interposed subsets of pixels and the dimming is applied to each subset in turn.

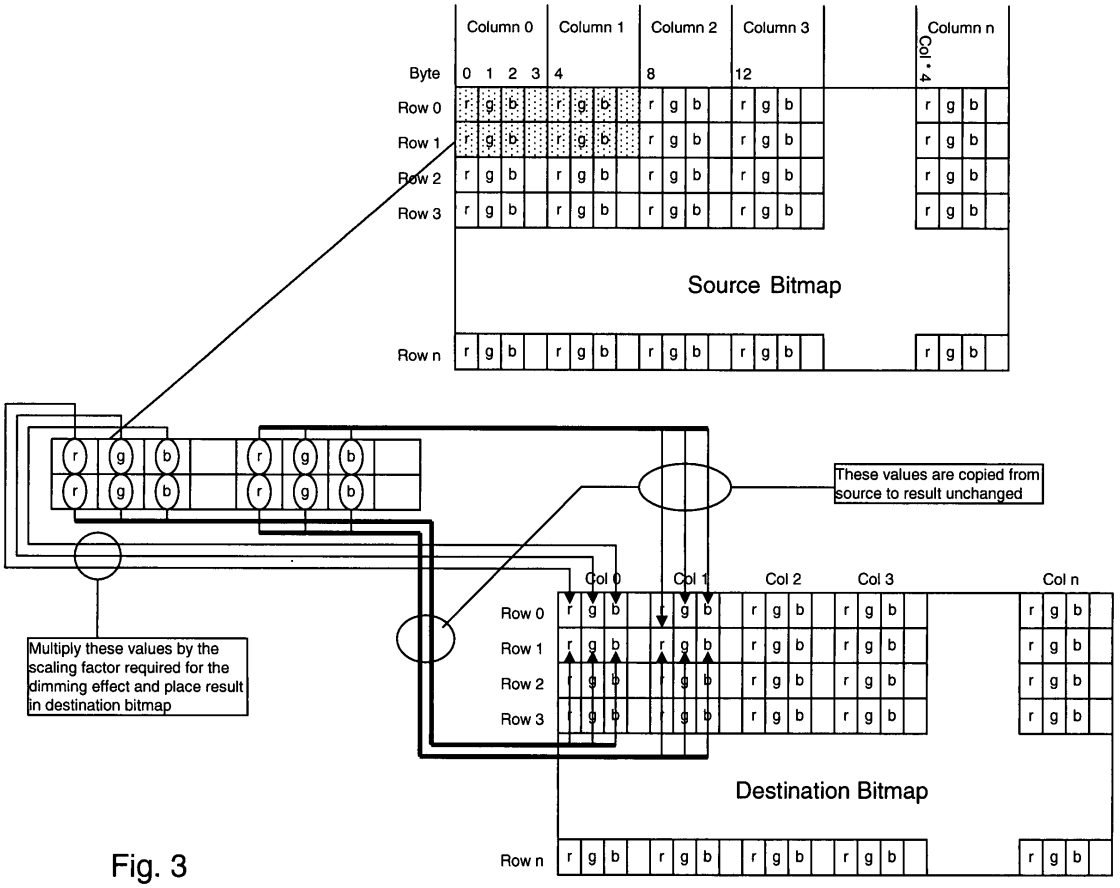

Fig. 3

EP 1 772 850 A1

Description

[0001] The present invention is directed to the field of display devices and the reduction in wear in such devices.

[0002] Computer display developments have resulted in large, bright screens using technologies such as Cathode Ray Tube (CRT), Plasma, Liquid Crystal Display (LCD) and Light Emitting Diode (LED), which can show any type of visual information in the form of characters or pictures. However, the working life of these displays is dependent on the type of image they are showing. More recent innovations in display technologies such as Organic LED (OLED) may also be prone to the problem.

[0003] Images with fixed areas where no change or movement take place over relatively long time periods, such as tables of data and graphs result in an effect generally called "burn-in". In cathode ray tube or plasma displays, burn-in takes the form of damage to the phosphor resulting from prolonged excitation. The role of the phosphor is to generate visible light and hence produce the image to be viewed. A damaged phosphor has a reduced ability to generate the visible light required leading to a fainter display. Burn-in manifests itself in a reduced ability of the display to show the expected picture at the required brightness and can be seen as a ghost of the static image that previously appeared. A good example of this can be seen in the television annunciations found on railway station platforms.

[0004] One area where the effects of burn-in could be particularly significant is in telephone Call Centres where the status of the Call Centre is shown on large displays to inform all the Call Centre staff of the situation regarding calls received, answered, waiting etc.

[0005] Figure 1 shows a typical screen shot of a call centre display. The display is used to show real-time statistics from a call centre controller. The information to be shown is grouped into display areas which can be as detailed or summarised as required. Up to 16 separate display areas can be configured on each screen. As values for items such as calls waiting or agents working cross thresholds, the affected item can be made to change colour. In Figure 1, specific blocks are shown where no change in the image presented will occur over long periods. These permanent or semi-permanent areas include the title bars (here labelled: "Agents", "Calls" and "Averages") and the on-screen BT logo.

[0006] Traditionally, the displays used by Call Centres use LED technologies, which show the least effects of burn-in relative to CRT, Plasma and LCD. However, the LED displays are expensive and inflexible, as they can show only a limited amount of information, and Call Centre managers are turning to the other display technologies to provide a more flexible solution to displaying the Call Centre status. Modern white-light LEDs are more prone to burn than earlier types.

[0007] CRT displays of a size sufficient to replace the LED displays used in Call Centres are unwieldy and extremely expensive. LCDs of similar size are more expensive still. Plasma displays of adequate size to replace LED displays are available and relatively affordable, and very flexible in that they are capable of displaying information in the form of static or moving pictures allowing their use for other purposes when the need to show Call Centre statistics is not present.

[0008] However, Plasma displays have the worst characteristics of all the technologies mentioned when it comes to burn-in.

[0009] Plasma displays are manufactured to conform to one of two broad specifications:

Industrial and Domestic. Industrial displays cost approximately two and a half times as much as the correspondingly sized Domestic displays, though their working life is five times as long.

[0010] Industrial displays are usually provided with internal features to increase their life by moving the displayed image across the screen by a small amount, effectively changing the colour or brightness of picture elements (pixels) by providing different information for them to display. This technique is described as Pixel Orbiting. However, although domestically rated displays are now beginning to be provided with this feature, it is not particularly effective when displaying images showing large areas of the same colour.

[0011] There have been various approaches to this problem. The problem of burn-in was discussed in New Scientist (21 March 2002) with reference to pixel-orbiting and the problem of fixed logos, as used by some television channels.:

http://www.newscientist.com/article.ns?id=dn2074

[0012] With pixel orbiting, the image is "orbited" or shifted around a display screen so as change the signal driving each pixel. The idea is that a high-intensity signal on one pixel will be replaced by a lower-intensity one, thus extending the life of the display. A hardware solution to plasma burn-in problem using pixel orbiting can be found at:

http://www.extron.com/technology/archive.asp?id=plasmab

[0013] Other solutions, such as designing images with no fixed areas and dimming the entire display screen, are discussed at:

http://www.scala.com/authoring/avoiding-plasma-burn-in.html
http://www.pioneerelectronics.com/pna/fag/detail/0.2076 149599173 149889639,00.html

[0014] US patent 6,313,878 suggests automatically dimming a section of the display when a high risk of burn-in is determined. The system works by monitoring the image displayed within a number of areas of the screen and dimming the display over any areas if the image in that area has not changed significantly over time. The result of this dimming can be highly visible to viewers resulting in a patchwork effect that significantly degrades the image displayed. US patent application 2003/0071769 suggests dimming each pixel in a plasma display one at a time. The result of this dimming is less likely to be visible to the viewer but the time taken to service the entire display will be much longer than the previous methods may not be able to cope with the very large displays now being produced.

[0015] Even if the image is continually changing, according to one or other of the above schemes, certain picture elements may be turned on at high intensity for a prolonged time leading to burn-in at those locations. Existing solutions using pixel orbiting require additional electronic circuitry; either incorporated in the body of the display unit or as an external addition to the display unit, and hence are costly to implement. The above references to the prior art are given for the purposes of providing background to the present invention and are not to be taken as an indication that the content of the cited prior art documents constitutes common general knowledge.

[0016] The present invention seeks to provide means for reducing wear in plasma and other displays caused by burn-in or similar cumulative destructive effects that is amenable to implementation in software and to incorporation into computer applications so as to avoid changes to existing displays. Its scope extends to any display technology subject to burn-in or similar degradations and its cost can be subsumed into that of the application making use of the invention. If required, for example for reasons of speed or economy, the invention may alternatively be implemented in hardware or in software as part of an improved display.

[0017] The present invention provides a method for reducing wear in a display device comprising a plurality of pixels including the steps of dividing the pixels into a plurality of subsets of interposed pixels; processing each subset in turn by reducing the intensity of the pixels in the subset and returning the pixels of the subset to normal intensity.

[0018] According to one aspect the invention provides the steps of returning each subset to normal intensity before reducing the intensity of the next subset. According to one aspect of the invention pixels adjacent in the display belong to different subsets.

[0019] The present invention provides a method for reducing wear in a display device comprising a plurality of pixels including the steps of dividing the pixels into a plurality of interposed subsets; processing the subsets by reducing the intensity of one pixel in each subset and returning the pixel to normal intensity. According to one aspect the invention a corresponding pixel in each of the subsets is processed in the same turn.

[0020] The present invention also provides a signal processing means operable to provide a signal to a display device for defining an image to be displayed; in which the signal processing means is operable for processing the signal to cause temporary reductions in the intensity of pixels in the displayed image according to a sequence in which the image comprises a plurality of interposed subsets of pixels and the sequence comprises each subset being reduced in intensity in turn.

[0021] The present invention provides a signal processing means operable to provide a signal to a display device for defining an image to be displayed; in which the signal processing device is operable for processing the signal to cause temporary reductions in the intensity of pixels in the displayed image according to a sequence in which the image comprises a plurality of subsets of interposed pixels; and the sequence comprises reducing the intensity of one pixel in each subset and returning the pixel to normal intensity. According to one aspect of the invention, the sequence comprises reducing the intensity of a corresponding pixel in each of the subsets in the same turn.

[0022] The invention achieves its goals by dimming or blanking individual picture elements (pixels) on the display device. According to one preferred aspect of the invention, about one quarter of the display's pixels are modified at any one time. As time progresses, the affected pixels are returned to full brightness and another set of one quarter of the pixels are dimmed or blanked. Ultimately, all the pixels on the display will have progressed through the bright-dim-bright cycle, and the process repeats until the application embodying the invention is terminated. The effect of this may be visible to the human eye depending on various parameters such as depth of attenuation (dimming) and time sequence. According to one aspect of the invention, controls are incorporated into the application to adjust the depth of dimming, to achieve a balance between minimising the visible effect and maximising the reduction in burn-in.

[0023] Various embodiments of the invention will now be described in more detail by way of example with reference to the drawings in which

Figure 1 shows a screen shot from a call centre display;

Figure 2 shows a call centre system with display, suitable for implementing the present invention;

Figures 3 to 6 show typical bitmap layouts and processing of pixels according to embodiments of the present invention;

Figures 7 and 8 show a displayed image to which the present invention has been applied.

[0024] The invention will be described with reference to BT call centres. BT call centres use IPCC Wallboard for displaying operating statistics to the staff working in the call centre. IPCC Wallboard is an application started when required either on a workstation that is dedicated to the displays or as a background task on a supervisor's machine. IPCC stands for IP Integrated Contact Distribution, a Cisco product name.

[0025] Figure 2 shows a typical call centre in block diagram form. A number of intelligent subsystems are shown linked by a common local area network (LAN). These intelligent subsystems include a plurality of work stations for the agents manning the telephones in the call centre. Each agent is provided with a conventional general-purpose personal computer and a telephone (here an IP phone interfacing to the LAN but in alternative arrangements conventional telephones may be used connecting into a conventional telephone switch). Each computer has a bus (not shown) to which are connected a central processing unit (not shown) the visual display unit, a keyboard, and a local memory (not shown). In the local memory are stored an operating system, a program for performing the call centre tasks, and storage areas for storing data (all not shown). Each computer has an interface to the LAN. The call centre LAN is also connected to the IPCC call centre and a call manager. Also connected to the LAN is a controlling workstation that may be a similar type of personal computer to the agents' workstation or, alternatively a more powerful model. The controlling workstation will comprise the same basic elements as the agents' computer but with additional video interface capability to drive one or more large display screens or wallboards.

[0026] In operation, data is extracted from the IPCC system by the controlling workstation and fed to the defined display areas, as illustrated in Figure 1. A number of display screens (typically up to 9) may be connected to the controlling workstation in addition to a standard computer monitor and each display screen can be configured to show a different set of items (more than one external screen usually requires additional video hardware to be fitted in the workstation.)

[0027] As values for displayed items such as "Calls Waiting" or "Agents Working" cross thresholds, the affected displayed item can change colour to indicate that an action is required. Each data item can have two thresholds defined. As the value of the item crosses these thresholds, it can change colour to indicate that some action is required. These thresholds are specific to each display area though they may be copied from one area to another to keep them in step.

[0028] The principle of operation of the invention according to a first aspect will now be described. According to this aspect, the image to be displayed is processed by a computer, typically the controlling workstation, that feeds the image information to one or more separate devices, e.g. plasma displays, for display. The image-processing computer could, conveniently, also be the originator of the image and the processing described below could conveniently be integrated into the software package that generates the data to be displayed.

[0029] The computer application implementing the invention prepares the image to be shown as a so-called source bitmap in computer memory. The source bit map will have an entry corresponding to each physical pixel in the display. Each entry will contain information of the intensity of the corresponding pixel. To better understand the implementation, the bitmap may be conceptually divided into cells, each containing four picture elements (pixels) in square formation.

[0030] The application scans this source bitmap pixel by pixel and determines whether each pixel in turn requires to be dimmed. Pixels not requiring to be dimmed are copied to the corresponding position in a destination bitmap of the same size as the source bitmap. Pixels that require dimming have their individual colour values, red, green and blue separated out and the colour brightness values multiplied by a theoretical multiplicand between 0 and 1. The resultant dimmed pixel is copied to the destination bitmap in a position corresponding to the one it occupied in the source bitmap.

[0031] To improve the speed of processing the pixel values, the multiplication described in the previous paragraph can be performed using fixed point arithmetic. To achieve this, the actual multiplicand is expressed as a numeric value between 0 and 255, with the value of 255 corresponding to full brightness and 0 corresponding to switching off the pixel completely. Scaling is carried out by dividing the result of the multiplication by 256 (which may be simply achieved by bit shifting) and truncating the result as necessary. For example, if the colour intensity values are 8 bit quantities, truncation will be to the range 0 to 255. The formula can be expressed as follows:

$$\text{Resultant colour intensity} = (\text{Original colour intensity} * \text{Dimming factor}) / 256$$

[0032] The resultant bitmap is copied to the display device in the usual way using the position and dimensions defined by the controlling application, i.e. is the position and size of the image on the display screen. Where the visual effect of pixel-dimming is critical, the reduction in intensity may be carried out in stages according to a further aspect of the invention. According to this aspect, the desired dimming is achieved in stages distributed across the dimming cycle with the theoretical multiplicand set to a different value between 0 and 1. For example, for a four-stage dimming process,

instead of multiplying once by 0.5, the processing can consist of four successive multiplications by 0.84. Fixed-point multiplication can then be achieved, as before, by using the equivalent actual multiplicand between 0 and 255, as before. The return to normal intensity is also achieved in stages. This can be achieved either by multiplication by an appropriate actual multiplicand greater than 1 or reuse of stored values. In this latter case, at each stage of the dimming process, the reduced intensity values for the affected pixels are stored in memory cross-referenced to the pixel identity and to the stage in the dimming process in which they are generated. In order to return the pixel to normal intensity, these stored values are simply read in order and written into the destination bitmap at the appropriate location.

[0033] According to another aspect of the present invention, the timing of changes to the selection of pixels for dimming can be achieved in a number of ways, including: synchronously with the updating of the information to be displayed (controlled by the application generating the information) and on a fixed cycle, asynchronously from any information change. If the application has a fixed cycle of updating the information to be displayed (as in the case of collecting real-time statistics), each update can be used to trigger the dimming action. Even if the information does not change from the previous cycle, a new bitmap will be generated by the application as the bitmap intended for display. The case where the application does not regularly refresh the display will require intervention by an asynchronous process to perform the display refresh. This can be implemented as a separate thread of execution to handle the display refresh. Arbitration will be required to regulate access to the source bitmap when the application "wakes up" to update the image at the same time that the display refresh process is processing the reading the source bitmap to apply the dimming effect. A number of conventional techniques for managing shared memory can be used to achieve shared access without corruption of the image displayed. According to a preferred embodiment, the processing is arranged so that each pixel in a display is processed at an interval of no more than five minutes.

[0034] Hence, according to one preferred aspect of the invention, the selection of pixels to which the dimming action is applied changes with each change to the image to be displayed According to another preferred aspect of the invention, the selection changes at regular intervals. According to a further preferred aspect of the invention, the selection changes at irregular intervals depending on the display characteristics.

[0035] A sequence in which pixels may be dimmed according to the present invention will now be described with reference to a row and column arrangement, i.e. a matrix of pixels as commonly found in dot matrix display systems, as illustrated in Figures 3 to 6. In such a matrix, the rows are commonly referenced by sequential numbering starting from one edge of the matrix. Similarly, the columns are referenced by sequential numbering starting from another edge of the matrix orthogonal to the first edge. For the initial selection, all pixels whose column and row have even values are affected. On the second selection, pixels with odd column values and even row values are affected. On the third selection, pixels with odd row values and even column values are selected. On the fourth selection, pixels with odd row values and odd column values are affected. The cycle then returns to the first selection and this continues as long as the controlling application is running. The actual order in which the pixels are dimmed may be changed from that given above without departing from the scope of the invention, for example the sequence of row and column values: even/even, even/odd, odd/odd, odd/even may be used

[0036] Preferably, the time interval of dimming each set of pixels is controlled independently of the display refresh rate or the image update rate so that the interval can be set with some degree of precision. Alternatively, if the application is expected to be displaying information changing slowly but fairly regularly, such as the statistics required in a Call Centre, then the update interval of the actual data can be used to determine the timing of the pixel selection, reducing the complexity in the system. For domestic plasma displays, an update interval of less than five minutes will be suitable. If, however, the update interval exceeds five minutes, the addition of an independent refresh timer for the selection interval should be implemented.

[0037] An alternative embodiment will now be described. As with the earlier embodiment, described above, the invention is applied by means of a computer application preparing the image to be displayed as a bitmap in computer memory. The bitmap is conceptually divided into cells containing four picture elements (pixels) in square formation.

[0038] A mask is constructed in am auxiliary bitmap of identical size to the source bit map by setting all the pixels in the auxiliary bitmap to grey (i.e. intermediate intensity) then drawing a grid of one pixel wide lines of white (i.e. maximum intensity) spaced one pixel apart both horizontally and vertically. When the mask is combined with the original bitmap, the effect is to dim pixels corresponding to the remaining grey pixels in the mask, and leave the other pixels at full brightness. The depth of the dimming effect can be altered by changing the actual colour of the grey pixels, black effectively blanks the pixel and full white disables the dimming functionality of the mask. The depth can be set by the operator, who adjusts the depth of the dimming effect to provide a balance between an intrusively visible effect and the requirement for prolonging the life of the display device.

[0039] The result of the combining of the source bitmap and the mask in the auxiliary bitmap is copied to the display device using the position and dimensions defined by the controlling application.

[0040] On the next occasion that the display is refreshed, the same process is performed except that the grid of white lines drawn on the mask bitmap is offset by one pixel horizontally or vertically. By alternating the axis of the offset, the pixel to be dimmed or blanked appears to rotate amongst four adjacent pixels in a square formation. This "rests" the

pixel that appears on the computer monitor for one period in four.

[0041] Figure 3 shows a typical layout of pixels in bitmap memory. The source bit map is shown at the top of the figure as comprising a matrix of rows and columns. The rows are here labelled from 0 (at the top of the bitmap) to n (at the bottom). Similarly, the columns are here labelled from 0 (at the left of the bitmap) to n (at the right), although image geometries other than square may be chosen. Each intersection of a row and a column in the bitmap is referred to as a "location". As the bitmap relates to a colour display, each location contains three pixel elements: red, green and blue. In a typical 8-bit implementation, each element will consist of one byte of memory. Binary addressing makes it easier to deal in memory areas that are formed from a power of two elements. Hence each location has a fourth element that is not used in this RGB display example.

[0042] The location of each pixel within the image (i.e. its memory address within the image array) is found by adding its column number multiplied by the number of elements defining a single pixel (including any unused elements, in this case four) to its row number multiplied by the width of the bitmap in bytes (i.e. the number of elements in each row). The width of the bitmap can be calculated by multiplying the number of columns by the number of elements defining a single pixel. The result of this addition is in turn added to the base address of the bitmap array in memory to give the base address of the pixel, i.e. the memory address of the first element of that pixel in the bitmap.

[0043] Variations on this addressing will depend on the colour depth chosen by the application developer, and the computer platform on which the bitmap is stored, in particular on the configuration of the memory. Rows in memory may be inverted, i.e. the row at the top of the bitmap may start at a low or high memory address. This is does not affect the workings of the present invention as it is the entire bitmap that is processed on each application of the dimming effect.

[0044] Figure 3 shows the copying of data from the source bitmap to the destination bitmap for the first operation in the four operation cycle. A subset of four pixels is shown greyed out in the source bitmap. The same four pixels are shown in more detail in the centre of the figure showing how only one of the pixels in this subset is dimmed before being copied (indicated by faint lines) to the destination bitmap. The remaining three pixels in this subset are copied without modification (indicated by bold lines) to the destination bitmap. This process is carried out for the first operation on each subset of four pixels across the entire bitmap. An alternative way of viewing the division of the bit map is to consider a group consisting of the first pixel in each of the above subsets of four adjacent pixels (which we may distinguish as the "local" subsets) to itself constitute a subset extending across the bit map (i.e. an "extended" subset). According to this view , each of the pixels in one of the local subsets constitute, together with the corresponding pixels in the others ones of the local subsets, an extended subset of pixels extending across the bitmap with the pixels of each extended subset being interposed with the pixels of the other three extended subsets. An illustration of one such extended subset may be seen in the regular array of dots (representing dimmed pixels) in the enlarged screen image of Figure 8.

[0045] Figure 4 shows the same source and destination bitmaps as Figure 3. Figure 4 shows the copying of data from the source bitmap to the destination bitmap for the second operation in the four operation cycle. The four pixels of the same local subset are again shown in more detail in the centre of the figure showing how a second one of the pixels in this subset is dimmed before being copied (indicated by faint lines) to the destination bitmap. The remaining three pixels in this subset are copied without modification (indicated by bold lines) to the destination bitmap. This process is carried out for the second operation on each local subset of four pixels across the entire bitmap.

[0046] Figure 5 shows the same source and destination bitmaps as Figures 3 and 4. Figure 5 shows the copying of data from the source bitmap to the destination bitmap for the third operation in the four operation cycle. The four pixels of the same local subset are again shown in more detail in the centre of the figure showing how a third one of the pixels in this subset is dimmed before being copied (indicated by faint lines) to the destination bitmap. The remaining three pixels in this subset are copied without modification (indicated by bold lines) to the destination bitmap. This process is carried out for the third operation on each local subset of four pixels across the entire bitmap.

[0047] Figure 6 shows the same source and destination bitmaps as Figures 3 to 5. Figure 6 shows the copying of data from the source bitmap to the destination bitmap for the fourth operation in the four operation cycle. The four pixels of the same local subset are again shown in more detail in the centre of the figure showing how a fourth and last one of the pixels in this subset is dimmed before being copied (indicated by faint lines) to the destination bitmap. The remaining three pixels in this subset are copied without modification (indicated by bold lines) to the destination bitmap. This process is carried out for the fourth operation on each local subset of four pixels across the entire bitmap.

[0048] To revisit the description of copying of pixels from the source to the destination bitmap in terms of the extended subsets, each pixel in a first extended subset, e.g. with reference to Figure 3, pixels in column 0, rows 0; 2, 4, etc; column 2, rows 0, 2, 4, etc and so on up to column n-1, row n-1 (assuming a matrix with an even number of rows and columns) is dimmed before being copied (indicated by faint lines) to the destination bitmap. The pixels of the other three extended subsets (basically the remainder of the bitmap) are copied without modification (indicated by bold lines) to the destination bitmap. With reference to Figure 4, pixels of the second extended subset, i.e. those in column 1, rows 0; 2, 4, etc; column 3, rows 0, 2, 4, etc and so on up to column n, row n-1 (assuming a matrix with an even number of rows and columns) is dimmed before being copied (indicated by faint lines) to the destination bitmap. The pixels of the other three extended subsets are copied without modification (indicated by bold lines) to the destination bitmap. With reference to Figure 5,

pixels of the third extended subset, i.e. those in column 0, rows 1, 3, 5, etc; column 2, rows 1, 3, 5, etc and so on up to column n-1, row n (assuming a matrix with an even number of rows and columns) is dimmed before being copied (indicated by faint lines) to the destination bitmap. The pixels of the other three extended subsets are copied without modification (indicated by bold lines) to the destination bitmap. With reference to Figure 6, pixels of the fourth and final extended subset, i.e. those in column 1, rows 1, 3, 5, etc; column 3, rows 1, 3, 5, etc and so on up to column n, row n (assuming a matrix with an even number of rows and columns) is dimmed before being copied (indicated by faint lines) to the destination bitmap. The pixels of the other three extended subsets are copied without modification (indicated by bold lines) to the destination bitmap.

[0049] It should be noted that the copying in each operation of three pixels in each subset without modification (as indicated by bold lines) from the source bitmap to the destination bitmap effectively restores to normal intensity whichever pixel was the subject of dimming in the previous operation..

[0050] A practical implementation of the invention will now be described in detail in the form of a simple software application designed to display a digital clock on a computer monitor.

[0051] This is a test application designed to illustrate an implementation of the PlasmaSaver burn-in reduction function. It is written using Borland Delphi® 2005, as a native Windows application, i.e. not making use of Microsoft .NET. The application is made up of a number of files with all the significant ones shown here. In this example, all the files should be copied to the same folder on a workstation having the Delphi® 2005 development environment installed. By careful translation of the source code, it should be possible to compile and run this application using other programming languages such as C, C++, C#, etc.

Clock.dpr

[0052] This is the start point for the application. It is usually maintained automatically by BDE.

--- File: Clock.dpr ---

```
program Clock;

// Identify user-defined units that make up the application. Note
that many
// units are included by implication as they are taken from BDE
run-time
// libraries.
uses
  Forms,
  MainForm in 'MainForm.pas' {Form1},
  SettingsDlg in 'SettingsDlg.pas' {SettingsDialog},
  DxPlasmaSaver in 'DxPlasmaSaver.pas' {PlasmaSaverSettings:
TFrame};

{$R *.res}  // Load resources required for the application to run.
If this file
            // (Clock.res in this case) is missing it can usually
be re-created
            // by BDE.
```

```
begin
  // This code is provided by the BDE and normally is not
modified. Its purpose
  // is to prepare the application for execution by initialising
the form
  // (window) definitions before they are displayed.
  //
  // TForm1 is the window that displays the clock.
  // TSettingsDialog is the window that allows the user to
experiment with the
  // PlasmaSaver settings.
  //
  // The statement Application.Run displays the primary form
(TForm1).
  //
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TSettingsDialog, SettingsDialog);
  Application.Run;
end.
```

--- End of File: Clock.dpr ---

DxPlasmaSaver.pas

[0053] This is the unit that embodies the burn-in reduction functionality.

--- File: DxPlasmaSaver.pas ---

```
// This unit embodies the PlasmaSaver burn-in reduction. It is
embedded in a
// Delphi class to encapsulate the code and make it more
transportable. The class
// TPlasmaSaverSettings defines a "composite control" which is
used to allow
// application users to alter the depth of the PlasmaSaver effect.
This control
// is embedded in a form or dialogue box to make it visible.

unit DxPlasmaSaver;

interface

// Make references to library units used by the application. All
are provided
// by Borland.
uses
  Windows, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, ComCtrls, StdCtrls;
```

```
type
  // The TPlasmaSaver class embodies the majority of the burn-in
reduction
  // function. All other code is written in support of this class.
It is defined
  // as a class to permit the possibility of adding functionality
by the usual
  // processes of object oriented languages.

  TPlasmaSaver = class(TObject)
  public
    //
    // Procedure: Apply
    // This procedure is called to apply the PlasmaSaver effect to
a bitmap
    // image and return the results of this in another bitmap. The
depth of the
    // effect can be controlled, as can the location of the first
pixel to be
    // affected.
    //
    class procedure Apply(
      aSrc: TBitmap;
          // Bitmap containing image prior to PlasmaSaver effect.

      aDest: TBitmap;
          // Bitmap containing image following application of
PlasmaSaver
          // effect.

      aDepth: Integer;
          // Depth of PlasmaSaver effect to apply. Value ranges
from 0
          // (no effect) to 255 (full effect). Values are clamped
so that out of
          // of range values do not cause a problem.

      aShifter: Integer
          // Position of first pixel to affect. The least
significant bit
          // controls horizontal motion and the next bit up
controls vertical
          // motion. Thus all possible positions of the first
pixel can be
          // encoded by integral values between 0 and 3 as
follows:
          //
          // aShifter Value   Pixel H   Pixel V
          // --------------   -------   -------
          //              0         0         0
          //              1         1         0
          //              2         0         1
          //              3         1         1
```

```
            //
    );

    //
    // Procedure: IncrementShifter
    // This procedure is used to determine the next pixel to be
affected by
    // rotating the affected position. This implementation of the
incrementer
    // simply increases the value of aShifter by one and rolls
values exceeding
    // 4 back to 0. Other schemes could have been implemented but
to little
    // effect. There is little difference in visual effect between
this simple
    // incrementing scheme which proceeds from top left through
top right,
    // bottom left and bottom right, when compared against a
"true" orbiting
    // scheme progressing through top left, top right, bottom
right and bottom
    // left.
    //
    class procedure IncrementShifter(
      var aShifter: Integer
        // Value to increment. Values resulting progress from 0
through 1, 2
        // and 3, back to 0. Initial values outside this range are
truncated so
        // that the result is always in range.
    );
  end;

type
  // TPlasmaSaverSettings is a composite control that allows the
application
  // user to set the depth of the PlasmaSaver effect when
incorporated into a
  // form or dialogue box. This is for test purposes, any real-
life
  // application could use a properly constructed component that
can be
  // distributed as part of a set of library functions.
  //
  TPlasmaSaverSettings = class(TFrame)
    checkboxUsePlasmaSaver: TCheckBox;
        // Set by user to indicate whether PlasmaSaver is enabled
or not.
        // Results of changing the state are reflected in the
preview
        // (imageSaverEffect).

    trackbarPlasmaSaverDepth: TTrackBar;
```

```
        // Set the depth of the PlasmaSaver effect. Results of
changing the
        // position on the trackbar are reflected in the preview
        // (imageSaverEffect).

    imageSaverEffect: TImage;
        // Preview of the state defined by the above two controls.

    staticTextDepth: TStaticText;
        // A label for the trackbar.

    staticTextEffect: TStaticText;
        // A label for the preview.

    procedure trackbarPlasmaSaverDepthChange(Sender: TObject);
        // This event occurs when the position of the Depth slider
is altered.

    procedure checkboxUsePlasmaSaverClick(Sender: TObject);
        // This event occurs on changing the state of the
checkbox.
  private
    { Private declarations }
    fPreviewImage: TBitmap;
        // This contains the image to which the PlasmaSaver effect
is to be
        // applied. In this implementation, it is a large capital
"A".

    fPreviewDithered: TBitmap;
        // This bitmap contains the result of applying the
PlasmaSaver effect to
        // fPreviewImage.

    function GetEffectEnabled: Boolean;
        // Returns the state of the Enabled checkbox. The value is
set or
        // retrieved as a public property defined below.

    procedure SetEffectEnabled(const Value: Boolean);
        // Set the state of the Enabled checkbox. The value is set
or
        // retrieved as a public property defined below.

    function GetEffectDepth: Integer;
        // Returns the Depth trackbar position. The value is set
or
        // retrieved as a public property defined below.

    procedure SetEffectDepth(const Value: Integer);
        // Sets the Depth trackbar position. The value is set or
        // retrieved as a public property defined below.
  public
```

```
    { Public declarations }
    constructor Create(AOwner: TComponent); override;
      // Standard constructor is overridden to allow construction
of local
      // variables.

    destructor Destroy; override;
      // Standard destructor is overridden to allow destruction of
local
      // variables.

    procedure ShowEffect;
      // Refresh the preview area. Required because this
implementation of
      // composite controls does not of itself have a Repaint
event that works
      // so refresh has to occur on a procedure call. A
commerically
      // distributable control will not have this problem.

    property EffectEnabled: Boolean read GetEffectEnabled write
SetEffectEnabled;
      // Get or set the state of the Enabled checkbox.

    property EffectDepth: Integer read GetEffectDepth write
SetEffectDepth;
      // Get or set the position of the Depth trackbar.
  end;

implementation

{$R *.dfm}

{ TPlasmaSaver }

class procedure TPlasmaSaver.Apply(aSrc, aDest: TBitmap; aDepth:
Integer;
  aShifter: Integer);
  //
  // This procedure embodies the PlasmaSaver burn-in reduction
function. The
  // effect is to take a
  // source bitmap, identified by aSrc and modify pixels in it by
adjusting the
  // brightness of pixels according to a defined pattern. The
pattern is altered
  // by using different values for aShifter on each call to the
procedure.
  // The modified bitmap can then be found in aDest, which, when
displayed,
  // shows the PlasmaSaver effect.
  //
const
```

```
  BlackPixel: RGBQUAD = ( rgbBlue: 0; rgbGreen: 0; rgbRed: 0;
rgbReserved: 0 );
var
  zRow:      Integer;       // An indexing variable for image rows.
  zCol:      Integer;       // An indexing variable for image
columns.
  zWidth:    Integer;       // Local copy of bitmap width.
  zHeight:   Integer;       // Local copy of bitmap height.
  zSrcPixels: pRGBQUAD;     // Pointer to memory array containing
bitmap.
  zDstPixels: pRGBQUAD;     // Likewise.
  zGrey:     Byte;          // Depth of effect to be applied to a
single pixel.
  zEffect:   Byte;          // Depth of effect to apply across the
entire bitmap.
  zOrbitRow: Integer;       // Holds the odd/even state for the row
of pixels
                            // currently being processed.
begin
  // Get width and height of destination bitmap
  zWidth := aDest.Width;
  zHeight := aDest.Height;

  with aDest.Canvas do
  begin
    if aDepth = 0 then
      // There is no manipulation work to do as the depth of the
effect to be
      // applied is zero. Simply copy the source bitmap to the
destination.
      Draw( 0, 0, aSrc )

    else
    begin
      // Clamp the depth value to a range of 0..255.
      if aDepth < 0 then
        aDepth := 0
      else if aDepth > 255 then
        aDepth := 255;

      // Calculate the multiplier value corresponding to the
depth. Minimum
      // effect is applied when zEffect has a value of 255,
maximum effect
      // occurs when zEffect has a value of 0. Cast the result as
a byte
      // (unsigned value) as aDepth is actually an integer.

      zEffect := Byte(255 - aDepth);

      // Clamp the range of aShifter to 0..3. The reason for this
is described
      // in the unit interface section above.
```

```
      aShifter := aShifter and 3;

      // For each row of pixels in the bitmap, calculate its
starting position
      // in memory. Although memory for the bitmap is contiguous,
scan lines
      // are rounded up to 4 byte boundaries. If pixels are
represented by
      // RGBTRIPLE structures (3 bytes in length), there may be
sequences of
      // unused bytes in memory, and the number of affected memory
locations and
      // their offsets cannot be directly calculated by simple
multiplications.
      // By using bitmaps containing RGBQUAD structures, the
alignment issue is
      // not present and all pixels are contiguous. However, in
case the
      // situation is changed by alterations in technology, use
the supplied
      // functions to find the start of each scan line.

      for zRow := 0 to zHeight - 1 do
      begin
        zSrcPixels := aSrc.ScanLine[zRow];    // First pixel in
source
        zDstPixels := aDest.ScanLine[zRow];   // First pixel in
destination

        // Get odd/even state of row, shifted left one bit to
align with bit
        // in zShifter.

        zOrbitRow := ((zRow and 1) shl 1);

        // For each pixel on the selected scan line, see if its
value is to be
        // copied unaltered or whether it is to be affected by the
PlasmaSaver
        // effect.

        for zCol := 0 to zWidth - 1 do
        begin

          // If the row position (odd/even) and column position
match the
          // corresponding bits in zShifter, the pixel is to be
modified by
          // dimming. Otherwise, the pixel should be shown at its
original
          // brightness.
```

```
            if ((zCol and 1) or zOrbitRow) = aShifter then
              zGrey := zEffect
            else
              zGrey := $FF; // Zero effect

            // To improve efficiency, isolate the special cases
    where a full
            // brightness pixel is to be placed, in which case it is
    simply
            // copied, and a zero brightness pixel in which case a
    black pixel
            // is to be copied. All intermediate values of grey
    require three
            // multiplications and three divisions, which are to be
    avoided if
            // possible.

            if zGrey = $FF then
              zDstPixels^ := zSrcPixels^

            else if zGrey = 0 then
              zDstPixels^ := BlackPixel

            else
            begin
              // For each colour in the destination pixel, multiply
    it by the
              // effect depth (a value from 0 to 255) with 0
    corresponding to
              // zero brightness and 255 to full brightness. This
    will result
              // in a value between 0 and 65025 which, when divided
    by 255
              // gives a colour intensity between 0 and 255. In
    theory, clamping
              // the result by anding it with 255 is unnecessary but
    is included
              // as a precaution against overflows. The fields
    rgb... can only
              // contain values between 0 and 255.

              zDstPixels^.rgbBlue  :=
                ((Integer(zSrcPixels^.rgbBlue)  * Integer(zGrey))
    div 255) and 255;
              zDstPixels^.rgbGreen :=
                ((Integer(zSrcPixels^.rgbGreen) * Integer(zGrey))
    div 255) and 255;
              zDstPixels^.rgbRed   :=
                ((Integer(zSrcPixels^.rgbRed)   * Integer(zGrey))
    div 255) and 255;
            end;
```

```
          // Once a pixel has been processed, move the pixel
pointers to the
          // next in the row.
          Inc(zSrcPixels);
          Inc(zDstPixels);
        end;
      end;
    end;
  end;
end;

class procedure TPlasmaSaver.IncrementShifter(var aShifter:
Integer);
begin
  //
  // Calculate the next value for aShifter. This is simply a
progression from
  // 0 through 1, 2 and 3 and back to 0 again. Other schemes have
been tried,
  // to make the affected pixel "orbit", but nothing could be
gained from this
  // added sophistication.
  //
  aShifter := (aShifter + 1) and 3;
end;

{ TPlasmaSaver }

constructor TPlasmaSaverSettings.Create(AOwner: TComponent);
begin
  // Required to set up the visible controls.
  inherited Create(AOwner);

  // Create and initialise a bitmap on which to draw the preview
image.
  fPreviewImage := TBitmap.Create;
  with fPreviewImage do
  begin
    Width  := imageSaverEffect.Width;
    Height := imageSaverEffect.Height;
    PixelFormat := pf32Bit;
  end;

  // Create a bitmap to hold the image with the PlasmaSaver effect
applied.
  fPreviewDithered := TBitmap.Create;
  with fPreviewDithered do
  begin
    Width  := fPreviewImage.Width;
    Height := fPreviewImage.Height;
    PixelFormat := pf32Bit;
  end;
```

```
    // Create a picture on the preview bitmap.
    with fPreviewImage.Canvas do
    begin
      with Brush do
      begin
        Color := clBlack;
        Style := bsSolid;
      end;

      with Pen do
      begin
        Color := clBlack;
        Width := 1;
        Style := psSolid;
      end;

      // Draw a solid black background
      Rectangle( 0, 0, fPreviewImage.Width, fPreviewImage.Height );

      // Create a large white font
      with Font do
      begin
        Color := clWhite;
        Name := 'Times New Roman';
        Size := 36;
      end;

      // Draw a large white "A" near the centre of the preview.
      TextOut( 8, 0, 'A' );
    end;
  end;

  destructor TPlasmaSaverSettings.Destroy;
  begin
    // Remove local resources.
    fPreviewImage.Free;
    fPreviewDithered.Free;
    inherited;
  end;

  procedure TPlasmaSaverSettings.ShowEffect;
  var
    zDepth: Integer;
  begin
    // Show the result of applying (or not) the PlasmaSaver effect.
  If the
    // Enabled checkbox is not checked, appy an effect with a depth
  of zero, i.e.
    // no effect.

    if checkboxUsePlasmaSaver.Checked then
      zDepth := EffectDepth
    else
```

```
    zDepth := 0;

  // Apply the PlasmaSaver effect to the preview bitmap, using a
depth defined
  // above. Don't bother with the Shifter parameter for the
preview. Other
  // versions of this control may show the effect of the Shifter
running.

  TPlasmaSaver.Apply(
    fPreviewImage,       // Source bitmap
    fPreviewDithered,    // Result of applying the PlasmaSaver
effect
    zDepth,              // Depth of effect to apply
    0                    // Shifter - not used here.
  );

  // Show the results of the PlasmaSaverEffect.
  imageSaverEffect.Canvas.Draw(0, 0, fPreviewDithered );
end;

procedure TPlasmaSaverSettings.checkboxUsePlasmaSaverClick(Sender:
TObject);
begin
  // Redraw the preview because the state of the Enabled checkbox
has changed.
  ShowEffect;
end;

procedure
TPlasmaSaverSettings.trackbarPlasmaSaverDepthChange(Sender:
TObject);
begin
  // Redraw the preview because the position of the Depth trackbar
has changed.
  ShowEffect;
end;

function TPlasmaSaverSettings.GetEffectEnabled: Boolean;
begin
  // Property access: Return the state of the Enabled checkbox.
  Result := checkboxUsePlasmaSaver.Checked;
end;

procedure TPlasmaSaverSettings.SetEffectEnabled(const Value:
Boolean);
begin
  // Property Access: Set the state of the Enabled checkbox, and
update the
  // preview to show any changes.
  checkboxUsePlasmaSaver.Checked := Value;
  ShowEffect;
end;
```

```
function TPlasmaSaverSettings.GetEffectDepth: Integer;
begin
  // Property Access: Get the depth of PlasmaSaver effect from the
trackbar
  // position. There are 32 steps on the trackbar and these are
translated to
  // a value between 0 and 255.
  Result := trackbarPlasmaSaverDepth.Position * 8;
  if Result > 255 then
    Result := 255;
end;

procedure TPlasmaSaverSettings.SetEffectDepth(const Value:
Integer);
begin
  // Property Access: Set the position of the Depth trackbar from
a value.
  // Check that the value is appropriate to the limits of the
trackbar. Update
  // the preview to reflect any new value set.
  with trackbarPlasmaSaverDepth do
  begin
    if Value < 0 then
      Position := 0
    else if Value > 248 then
      Position := 255
    else
      Position := Value div 8;
  end;

  ShowEffect;
end;

end.
```

--- End of File: DxPlasmaSaver.pas ---

MainForm.pas

[0054] This is the visible element of the application. It defines the main form for the application and a space on which to draw the clock face.

--- File: MainForm.pas ---

```
// This unit is the source code for the main form, used to display
the running
// clock. As this is a test application, a number of possible
optimisations
// have not been implemented.

unit MainForm;
```

```
interface

// Make references to library units used by the application. All
except the
// unit DxPlasmaSaver are provided by Borland.
uses
  Windows, SysUtils, Graphics, Forms, Menus, Classes, Controls,
ExtCtrls,
  DxPlasmaSaver;

type
  TForm1 = class(TForm)
    mainMenu: TMainMenu;                  // Menu bar at form top.
    menuItemClock: TMenuItem;             // Item on menu bar.
    menuItemClockExit: TMenuItem;         // Sub-item of
menuItemClock.
    menuItemClockSettings: TMenuItem;     // Sub-item of
menuItemClock.
    paintboxClock: TPaintBox;             // Area in form to paint
"PlasmaSaver"
                                          // affected image.
    timer_1Sec: TTimer;                   // One second cycle timer
to repeatedly
                                          // refresh the form.

    procedure FormDestroy(Sender: TObject);
        // Clean up when the form is destroyed. Definition can be
found
        // underneath the "implementation" statement below.

    procedure FormCreate(Sender: TObject);
        // Prepare some local variables before form is opened.

    procedure timer_1SecTimer(Sender: TObject);
        // Executed every second to display the current time.

    procedure menuItemClockExitClick(Sender: TObject);
        // Executed when the Exit menu item is clicked.

    procedure menuItemClockSettingsClick(Sender: TObject);
        // Executed to open a dialogue form to allow modification
of PlasmaSaver
        // settings.
  private
    { Private declarations }

    // The following five declarations form part of the
PlasmaSaver burn-in reduction function.

    bitmapClockImage: TBitmap;
        // An auxilliary bitmap where the displayed picture is
assembled.
```

```
    bitmapClockDithered: TBitmap;
        // An auxilliary bitmap where the PlasmaSaver effect is
applied before
        // display.

    Shifter: Integer;
        // Value altered every display cycle to choose which pixel
is to be
        // affected.

    EffectEnabled: Boolean;
        // When True, the PlasmaSaver effect is displayed. When
False, the
        // picture will be displayed at full brightness.

    EffectDepth: Integer;
        // Determines the depth of the PlasmaSaver effect to be
applied. A value
        // of zero means no effect, a value of 8 or greater
applies the full
        // effect. The effect depth is otherwise proportional to
this value.
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

uses
  SettingsDlg;  // Reference to the unit containing the definition
of the
                // Settings dialogue.

{$R *.dfm}      // Reference to an auxilliary file containing
information about
                // size and position of form elements. This is
usually maintain-
                // ed by the BDE.

procedure TForm1.menuItemClockSettingsClick(Sender: TObject);
begin
  // Display the Settings dialogue, passing and returning values
in
  // EffectEnabled and EffectDepth. If the Cancel button in the
Settings
  // dialogue is clicked, the values returned in EffectEnabled and
EffectDepth
  // remain unchanged.
```

```
  SettingsDialog.Execute(EffectEnabled, EffectDepth);
end;

procedure TForm1.menuItemClockExitClick(Sender: TObject);
begin
  // Instruct the current form to close. Because this is the main
form in
  // the application, the application stops running.
  Close;
end;

procedure TForm1.timer_1SecTimer(Sender: TObject);
// The content of this procedure is one part of the PlasmaSaver
burn-in reduction function.
// This procedure is executed every 1000 milliseconds so that
//
//    a)  a new time string is generated,
//
//    b)  The 'top left' pixel to be affected by the PlasmaSaver
burn-in reduction function
//        is changed.
//
// The position of the pixel is held in the variable Shifter,
which is updated
// on every cycle by calling a procedure in the TPlasmaSaver class
to provide
// the next value.
var
  zTimeStr: string;    // A string to contain a textual
representation of the
                       // current time.
  zSize: TSize;        // Dimensions of the time string created
above.
  zX, zY: Integer;     // Coordinates in bitmap where string is to
be written.
begin
  // Ensure image has same dimensions as displayable region of
form.
  bitmapClockImage.Width  := paintBoxClock.Width;
  bitmapClockImage.Height := paintBoxClock.Height;

  // Likewise, potentially resize the "dithered" image.
  bitmapClockDithered.Width  := paintBoxClock.Width;
  bitmapClockDithered.Height := paintBoxClock.Height;

  with bitmapClockImage.Canvas do
  begin
    // Prepare to paint the background of the image.
    Brush.Color := clBlack;    // Fill with black.
    Pen.Color := clBlack;      // Set the shape border to black as
well.

    // Draw a black rectangle the same size as the image.
```

```
    Rectangle(0, 0, bitmapClockImage.Width,
bitmapClockImage.Height);

    // Draw white characters on a black background.
    Font.Color := clWhite;

    // Create a string representation of the current time.
    zTimeStr := FormatDateTime('hh:nn:ss', Now);

    // Empirically set the size of the font to use so that the
text fits in
    // the displayable rectangle. There are more sophisticated
ways to do this
    // but for test purposes this is adequate.
    Font.Height := bitmapClockImage.Width div 4;

    // Calculate the dimensions of the text string showing the
time.
    zSize := TextExtent(zTimeStr);

    // Calculate the offsets required to centre the text string in
the disply-
    // able rectangle.
    zX := (bitmapClockImage.Width - zSize.cx) div 2;
    zY := (bitmapClockImage.Height - zSize.cy) div 2;

    // Draw the text in the bitmap called bitmapClockImage.
    TextOut(zX, zY, zTimeStr);
  end;

  // If the PlasmaSaver effect is enabled, it has to be applied.
However, if
  // the effect is not enabled, simply draw the image in
bitmapClockImage.
  if EffectEnabled then
  begin
    // The definition of this procedure can be found in the unit:
DxPlasmaSaver.
    TPlasmaSaver.Apply(
      bitmapClockImage,        // Image to modify.
      bitmapClockDithered,     // Result of PlasmaSaver effect.
      EffectDepth,             // Depth of effect (0=none,
8=maximum).
      Shifter                  // Position of top left pixel to
modify. Value
                               // is returned in this variable to
indicate pixel
                               // for the next cycle.
    );


    // Draw the modified bitmap to the form.
    paintBoxClock.Canvas.Draw(0,0, bitmapClockDithered);
```

```
    // Determine the position of the next top left pixel to
affect.
    TPlasmaSaver.IncrementShifter(Shifter);
  end
  else
    // PlasmaSaver effect not required so just show image
unmodified.
    paintBoxClock.Canvas.Draw(0,0, bitmapClockImage);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  // Create the necessary variables for the PlasmaSaver effect.
They have
  // been described adequately in the form definition above.
  bitmapClockImage     := TBitmap.Create;

  // Force the image to have a specific representation in memory.
  bitmapClockImage.PixelFormat := pf32Bit;

  bitmapClockDithered  := TBitmap.Create;

  // Force the image to have a specific representation in memory.
  bitmapClockDithered.PixelFormat := pf32Bit;

  // When the display first appears, apply PlasmaSaver to maximum
effect.
  EffectEnabled := True;
  EffectDepth := 255;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  // Clear space occupies by variables and return to the system.
  bitmapClockDithered.Free;
  bitmapClockImage.Free;
end;

end.
```

--- End of File: MainForm.pas ---

SettingsDlg.pas

[0055] This unit describes a dialogue box that is used to control the depth of the PlasmaSaver effect on the main form.

--- File: SettingsDlg.pas ---

```
// This unit is the source code for the Settings dialogue box. It
makes use of
```

```
// a "composite control" defined in the DxPlasmaSaver unit to give
a suitable
// user interface to control the depth of the PlasmaSaver effect,
with a visual
// preview.

unit SettingsDlg;

interface

// Make references to library units used by the application. All
except the
// unit DxPlasmaSaver are provided by Borland.
uses
  Windows, Classes, Graphics, Controls, Forms, Dialogs, ComCtrls,
StdCtrls,
  DxPlasmaSaver;

type
  TSettingsDialog = class(TForm)
    buttonOK: TButton;
        // Button used to close dialogue box and indicate that any
changed
        // values are to be accepted by the caller.

    buttonCancel: TButton;
        // Button used to close dialogue box and indicate that any
changed
        // values are to be ignored by the caller.

    plasmaSaverControl: TPlasmaSaverSettings;
        // A "composite control containing a slider, checkbox and
paintbox so
        // that the effect can be previewed.

    procedure FormPaint(Sender: TObject);
        // Called when the form must be repainted to ensure that
plasmaSaver-
        // Control is properly refreshed.

  private
    { Private declarations }
  public
    { Public declarations }

    // Function: Execute
    //
    // This function is called to
    //
    //    a)  Initialise the dialogue box with passed-in values,
    //
    //    b)  Display the dialogue box and accept user input to
adjust the
```

```
    //        PlasmaSaver.
    //
    //    c)  Return any modified values to the caller, indicating
whether the
    //        Cancel button was used.
    //
    function Execute(
      var EffectEnabled: Boolean; {in,out}
          // Gets or sets the state of the Enabled checkbox. This
has a side-
          // effect in plasmaSaverControl. This value is returned
unchanged
          // if the Cancel button is pressed.

      var EffectDepth: Integer {in,out}
          // Gets or sets the depth of the PlasmaSaver effect.
This has a side-
          // effect in plasmaSaverControl. This value is returned
unchanged
          // if the Cancel button is pressed.

    ): Boolean;
          // Returns True if the OK button was pressed.
          // Returns False if the Cancel button was pressed.
  end;

var
  SettingsDialog: TSettingsDialog;

implementation

{$R *.dfm}  // Load resources required to display the form.

{ TSettingsDialog }

function TSettingsDialog.Execute(var EffectEnabled: Boolean;
  var EffectDepth: Integer): Boolean;
begin
  // Copy the initial values to plasmaSaverControl to set up the
preview area in
  // the control.
  plasmaSaverControl.EffectEnabled := EffectEnabled;
  plasmaSaverControl.EffectDepth := EffectDepth;

  // Display the form as a modal dialogue box. The effect of
executing
  // ShowModal is, in this instance either mrOK or mrCancel. The
caller will
  // want to know which button was pressed and a boolean result is
adequate.
  Result := (ShowModal = mrOK);

  if Result then
```

```
  begin
    // Because the OK button was pressed, copy the current values
from
    // plasmaSaverControl. It doesn't make any difference to the
result if
    // the values are unchanged though, in some implementations,
it may be
    // more suitable to change Result to False if no changes to
the values
    // occurred.

    EffectEnabled := plasmaSaverControl.EffectEnabled;
    EffectDepth := plasmaSaverControl.EffectDepth;
  end;
end;

procedure TSettingsDialog.FormPaint(Sender: TObject);
begin
  // To ensure the preview is properly updated, trap the Form
Repaint request
  // and instruct the control accordingly.

  plasmaSaverControl.ShowEffect;
end;

end.
```

--- End of File: SettingsDlg.pas ---

[0056] The following files are created automatically by the Borland Development Environment (BDE) and do not need modifying by hand.

DxPlasmaSaver.dfm

[0057] Resource definitions for the "composite control" used to allow application operators control over the PlasmaSaver effect. Maintained solely by the BDE.

--- File: DxPlasmaSaver.dfm ---

```
object PlasmaSaverSettings: TPlasmaSaverSettings
  Left = 0
  Top = 0
  Width = 213
  Height = 85
  HorzScrollBar.Range = 213
  HorzScrollBar.Visible = False
  VertScrollBar.Range = 85
  AutoScroll = False
  Constraints.MaxHeight = 85
  Constraints.MaxWidth = 213
  Constraints.MinHeight = 85
```

```
    Constraints.MinWidth = 213
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -11
    Font.Name = 'Tahoma'
    Font.Style = []
    ParentFont = False
    TabOrder = 0
    TabStop = True
    object imageSaverEffect: TImage
      Left = 156
      Top = 24
      Width = 53
      Height = 53
    end
    object checkboxUsePlasmaSaver: TCheckBox
      Left = 8
      Top = 4
      Width = 89
      Height = 17
      Caption = 'Enabled'
      TabOrder = 0
      OnClick = checkboxUsePlasmaSaverClick
    end
    object trackbarPlasmaSaverDepth: TTrackBar
      Left = 0
      Top = 44
      Width = 150
      Height = 33
      Max = 32
      TabOrder = 1
      OnChange = trackbarPlasmaSaverDepthChange
    end
    object staticTextDepth: TStaticText
      Left = 8
      Top = 24
      Width = 33
      Height = 17
      Caption = 'Depth'
      TabOrder = 2
    end
    object staticTextEffect: TStaticText
      Left = 156
      Top = 4
      Width = 33
      Height = 17
      Caption = 'Effect'
      TabOrder = 3
    end
  end
--- End of File: PlasmaSaver.dfm ---
```

MainForm.dfm

[0058] Resource definitions for the application's main form.

```
--- File: MainForm.dfm ---
object Form1: TForm1
  Left = 0
  Top = 0
  Width = 347
  Height = 300
  Caption = 'Clock'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'Tahoma'
  Font.Style = []
  Menu = mainMenu
  OldCreateOrder = False
  OnCreate = FormCreate
  OnDestroy = FormDestroy
  PixelsPerInch = 96
  TextHeight = 13
  object paintboxClock: TPaintBox
    Left = 0
    Top = 0
    Width = 339
    Height = 266
    Align = alClient
  end
  object mainMenu: TMainMenu
    Left = 40
    Top = 40
    object menuItemClock: TMenuItem
      Caption = 'Clock...'
      object menuItemClockSettings: TMenuItem
        Caption = 'Settings'
        OnClick = menuItemClockSettingsClick
      end
      object menuItemClockExit: TMenuItem
        Caption = 'Exit'
        OnClick = menuItemClockExitClick
      end
    end
  end
  object timer_1Sec: TTimer
    OnTimer = timer_1SecTimer
    Left = 76
    Top = 40
  end
end
--- End of File: MainForm.dfm ---
```

SettingsDlg.dfm

[0059] Resource definitions for the dialogue box giving the application user the opportunity to change the depth of the PlasmaSaver effect.

--- File: SettingsDlg.dfm ---

```
object SettingsDialog: TSettingsDialog
  Left = 0
  Top = 0
  BorderStyle = bsDialog
  Caption = 'Settings'
  ClientHeight = 125
  ClientWidth = 225
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'Tahoma'
  Font.Style = []
  OldCreateOrder = False
  Position = poMainFormCenter
  OnPaint = FormPaint
  PixelsPerInch = 96
  TextHeight = 13
  object buttonOK: TButton
    Left = 60
    Top = 96
    Width = 75
    Height = 25
    Caption = 'OK'
    Default = True
    ModalResult = 1
    TabOrder = 1
  end
  object buttonCancel: TButton
    Left = 144
    Top = 96
    Width = 75
    Height = 25
    Cancel = True
    Caption = 'Cancel'
    ModalResult = 2
    TabOrder = 2
  end
  inline plasmaSaverControl: TPlasmaSaverSettings
    Left = 4
    Top = 4
    Width = 213
    Height = 85
    HorzScrollBar.Range = 213
```

```
      HorzScrollBar.Visible = False
      VertScrollBar.Range = 85
      AutoScroll = False
      Constraints.MaxHeight = 85
      Constraints.MaxWidth = 213
      Constraints.MinHeight = 85
      Constraints.MinWidth = 213
      Font.Charset = DEFAULT_CHARSET
      Font.Color = clWindowText
      Font.Height = -11
      Font.Name = 'Tahoma'
      Font.Style = []
      ParentFont = False
      TabOrder = 0
      TabStop = True
    end
  end
--- End of File: SettingsDlg.dfm ---
```

[0060] Figure 7 shows a typical display screen from a call centre display in which various statistical data relating to the operation of the call centre is displayed. The data is essentially similar to that shown in Figure 1, although the image here differs from that of Figure 1 in layout and details. The present invention is not restricted to any particular design of image and is equally applicable to both images. As can be seen from Figure 7, the invention can be implemented without a significant visual impact. The visual impact of applying the invention is better demonstrated in Figure 8 which shows a magnified section of the display of Figure 7. From Figure 8 it can be seen that a regular pattern of dimmed pixels is visible as a result of the selective dimming of pixels distributed across the display area. The strength of this visible effect may be controlled, as explained above.

[0061] The present invention is advantageously adapted for implementation in software. It can be incorporated into computer applications without any changes to the hardware or firmware of existing displays. Its scope extends to any display technology subject to burn-in problems and is cheap to implement as its cost is subsumed into that of the application making use of the invention. Alternatively, the present invention may be integrated in to a display device in the form of software or firmware or, where the advantages justify the additional cost, in hardware. Other embodiments of the invention will occur to those skilled in display technology without requiring any inventive activity on their part and are included within the scope of the invention.

[0062] According to a preferred embodiment, the present invention is particularly adapted for an efficient implementation in software by arranging that all processing of pixel intensity is achieved without any division operation (except simple bit-shifting) and with a single multiplication operation. The present invention is neither restricted to treating pixels conceptually divided into blocks of four pixels nor to pixel blocks of a square form but applies to other divisions of an image of various pixel counts and geometrical forms. The present invention is not restricted to masking pixels with lines spaced by a single pixel but may be implemented with other spacings to alter the visual effect. The preservation technique has application to both colour displays and monochrome. With monochrome displays, the processing will typically apply to a single intensity value per pixel. With colour, the processing will apply to each individual colour element, e.g. the red, green blue of conventional plasma displays, the four coloured LED groups used in some LED displays and other configurations.

[0063] As will be understood by those skilled in the art, the invention may be implemented in software, any or all of which may be contained on various transmission and/or storage media such as a optical, semiconductor or magnetic disc, tape or device so that the program can be loaded onto one or more general purpose computers or could be downloaded over a computer network using a suitable transmission medium.

Claims

1. A method for reducing wear in a display device comprising a plurality of pixels including the steps of dividing the pixels into a plurality of subsets of interposed pixels; processing each subset in turn by reducing the intensity of the

pixels in the subset and returning the pixels of the subset to normal intensity.

2. A method, as claimed in claim 1 including the steps of returning each subset to normal intensity before reducing the intensity of the next subset.

3. A method, as claimed in claim 1 or claim 2, in which pixels adjacent in the display belong to different subsets.

4. A method, as claimed in any above claim, in which no reduced-intensity pixels are adjacent to one another in the display.

5. A method, as claimed in any above claim including the steps of processing an image file comprising a plurality of entries with one entry for setting the intensity of each pixel in the display, the steps including selecting those entries that correspond to the pixels to be reduced in intensity and operating on those entries to effect a reduction in intensity of the corresponding pixel.

6. A method, as claimed in claim 5 in which each entry in the image file comprises one or more numbers defining the intensity of the corresponding pixel in the display, the processing including multiplying each number by a dimming factor.

7. A method, as claimed in claim 5 in which each entry in the image file comprises one or more numbers defining the intensity of the corresponding pixel in the display, the processing including combining the image file with a mask file comprising a first set of pixels set to maximum intensity and a second set of pixels set to a lower intensity.

8. A method, as claimed in claim 7 in which the first set of pixels form a grid and the second set of pixels occupy the spaces in the grid.

9. A method, as claimed in any above claim in which the intensity of each pixel is represented by a plurality of numbers, each number representing the intensity of a different colour component of the pixel.

10. A method, as claimed in any above claim including initiating a reduction in pixel intensity in response to a change in the image content.

11. A computer program or suite of computer programs for use with one or more computers to carry out the method as set out in any above claim.

12. A signal processing means operable to provide a signal to a display device for defining an image to be displayed; in which the signal processing means is operable for processing the signal to cause temporary reductions in the intensity of pixels in the displayed image according to a sequence in which the image comprises a plurality of subsets of interposed pixels and the sequence comprises each subset being reduced in intensity in turn.

13. A signal processing means as claimed in claim 12 including the steps of returning each subset to normal intensity before reducing the intensity of the next subset.

14. A signal processing means as claimed in any one of claims 12 or 13, in which pixels adjacent in the display belong to different subsets.

15. A signal processing means as claimed in any one of claims 12 to 14 , in which no reduced-intensity pixels are adjacent to one another in the display.

16. A signal processing means as claimed in any one of claims 12 to 15 including the steps of processing an image file comprising a plurality of entries with one entry for setting the intensity of each pixel in the display, the steps including selecting those entries that correspond to the pixels to be reduced in intensity and operating on those entries to effect a reduction in intensity of the corresponding pixel.

17. A signal processing means as claimed in claim 16 in which each entry in the image file comprises one or more numbers defining the intensity of the corresponding pixel in the display, the processing including multiplying each number by a dimming factor.

18. A signal processing means as claimed in claim 16 in which each entry in the image file comprises one or more numbers defining the intensity of the corresponding pixel in the display, the processing including combining the image file with a mask file comprising a first set of pixels set to maximum intensity and a second set of pixels set to a lower intensity.

19. A signal processing means as claimed in claim 18 in which the first set of pixels form a grid and the second set of pixels occupy the spaces in the grid.

20. A signal processing means as claimed in any one of claims 12 to 19 in which the intensity of each pixel is represented by a plurality of numbers, each number representing the intensity of a different colour component of the pixel.

21. A signal processing means as claimed in any one of claims 12 to 20 including initiating a reduction in pixel intensity in response to a change in the image content.

22. A display device incorporating the signal processing means as claimed in any of claims 12 to 21.

23. A computer program or suite of computer programs for use with one or more processing devices to provide any of the apparatus as set out in any one of claims 12 to 22.

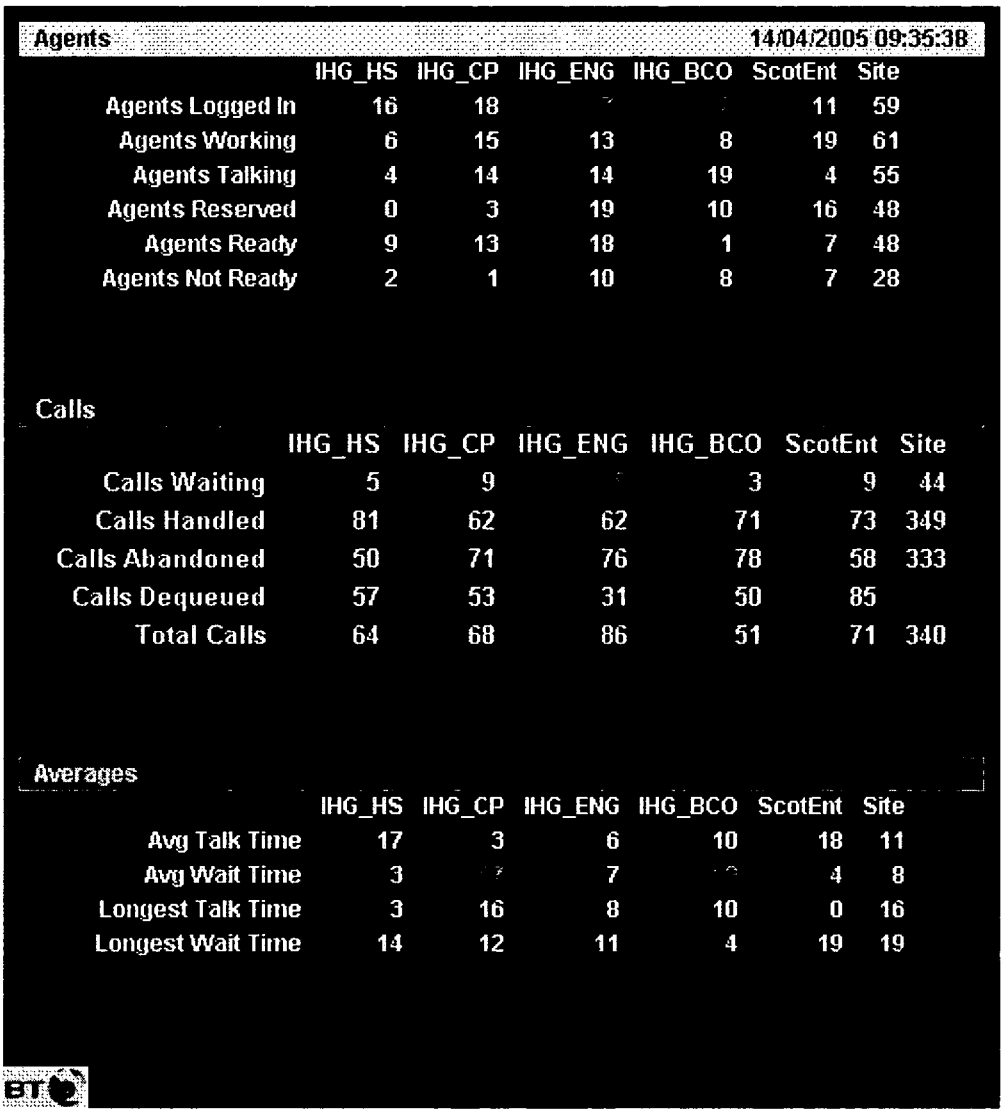
Agents
14/04/2005 09:35:38
IHG_HS IHG_CP IHG_ENG IHG_BCO ScotEnt Site
Agents Logged In 16 18 11 59
Agents Working 6 15 13 8 19 61
Agents Talking 4 14 14 19 4 55
Agents Reserved 0 3 19 10 16 48
Agents Ready 9 13 18 1 7 48
Agents Not Ready 2 1 10 8 7 28
Calls
IHG_HS IHG_CP IHG_ENG IHG_BCO ScotEnt Site
Calls Waiting 5 9 3 9 44
Calls Handled 81 62 62 71 73 349
Calls Abandoned 50 71 76 78 58 333
Calls Dequeued 57 53 31 50 85
Total Calls 64 68 86 51 71 340
Averages
IHG_HS IHG_CP IHG_ENG IHG_BCO ScotEnt Site
Avg Talk Time 17 3 6 10 18 11
Avg Wait Time 3 7 4 8
Longest Talk Time 3 16 8 10 0 16
Longest Wait Time 14 12 11 4 19 19

Fig. 1

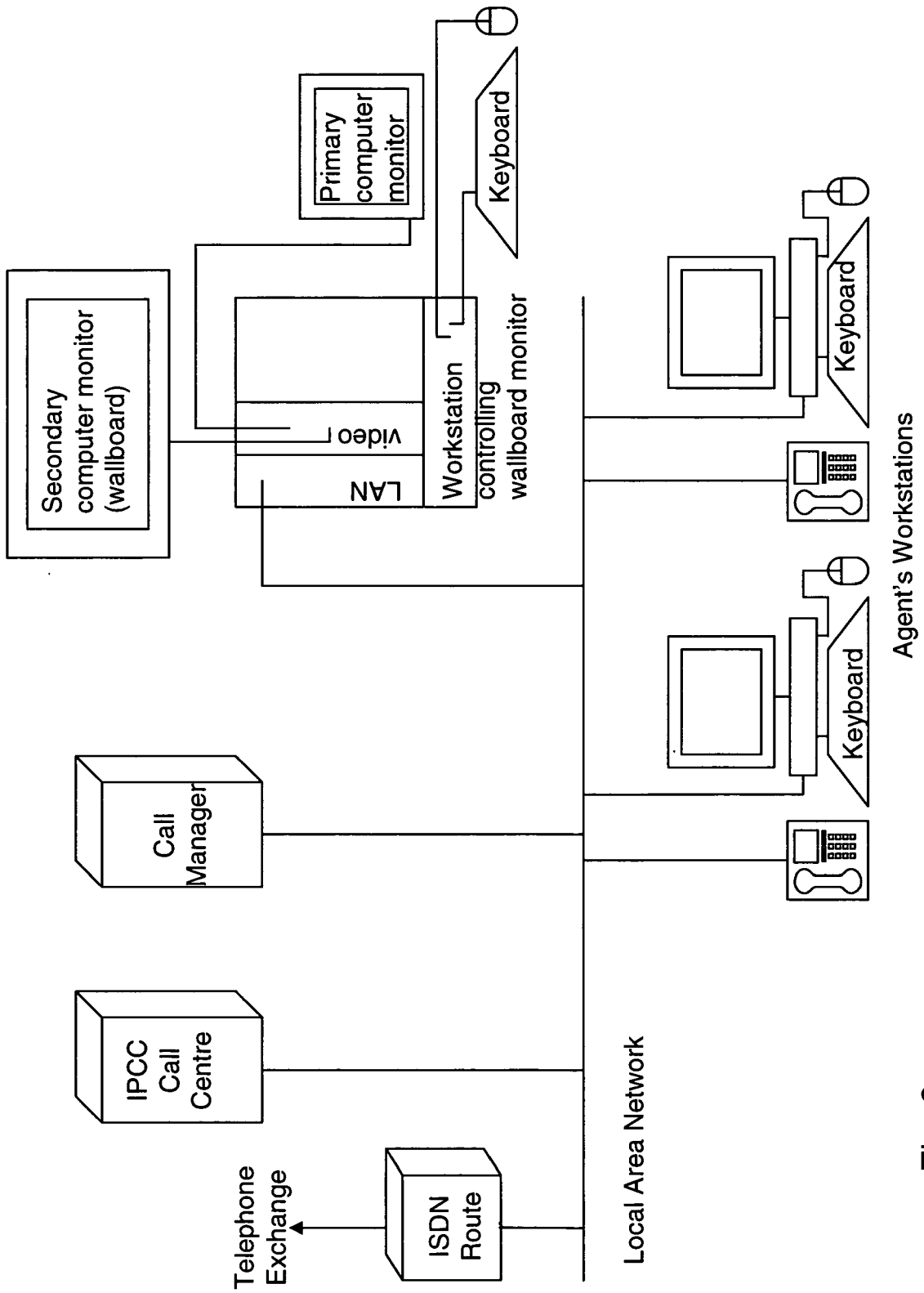
Secondary computer monitor (wallboard)
Primary computer monitor
Keyboard
Workstation controlling wallboard monitor
LAN
Video
Call Manager
IPCC Call Centre
ISDN Route
Telephone Exchange
Local Area Network
Keyboard
Keyboard
Agent's Workstations

Fig. 2

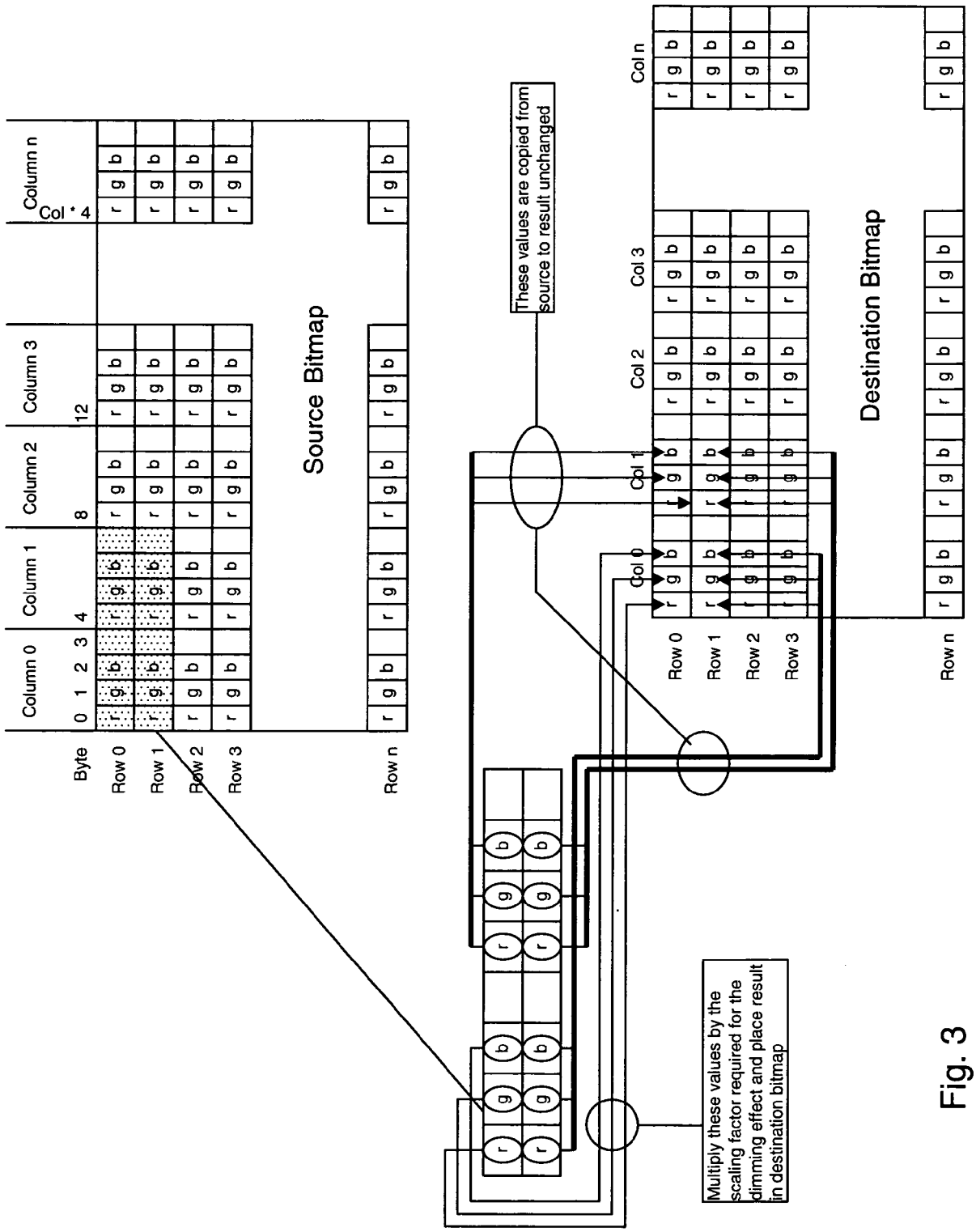
Source Bitmap
Destination Bitmap
These values are copied from source to result unchanged
Multiply these values by the scaling factor required for the dimming effect and place result in destination bitmap
Byte
Row 0
Row 1
Row 2
Row 3
Row n
Column 0
Column 1
Column 2
Column 3
Column n
Col * 4
Col 0
Col 1
Col 2
Col 3
Col n

Fig. 3

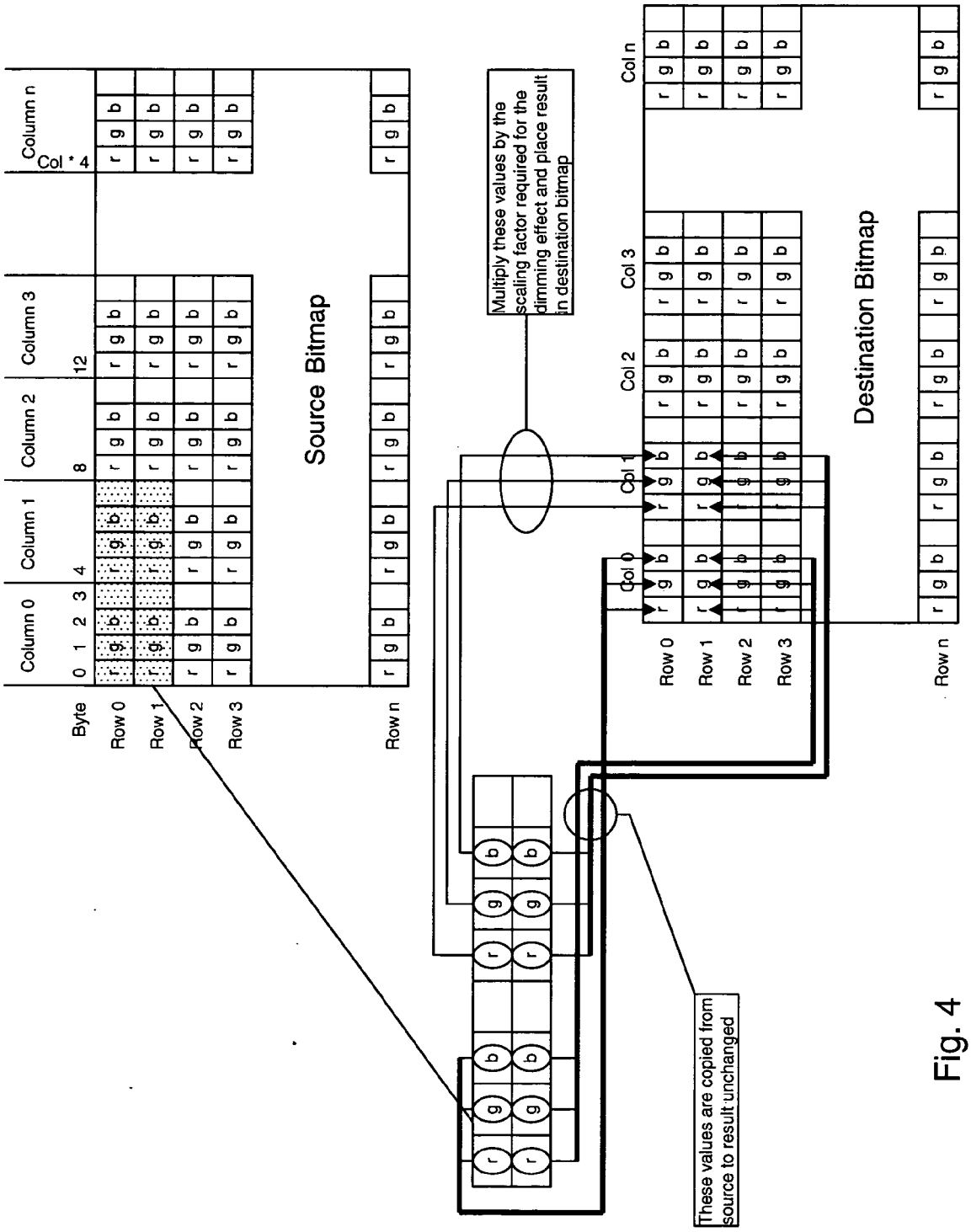
Source Bitmap
Destination Bitmap
Multiply these values by the scaling factor required for the dimming effect and place result in destination bitmap
These values are copied from source to result unchanged
Column 0
Column 1
Column 2
Column 3
Column n
Col * 4
Byte
Row 0
Row 1
Row 2
Row 3
Row n
Col 0
Col 1
Col 2
Col 3
Col n

Fig. 4

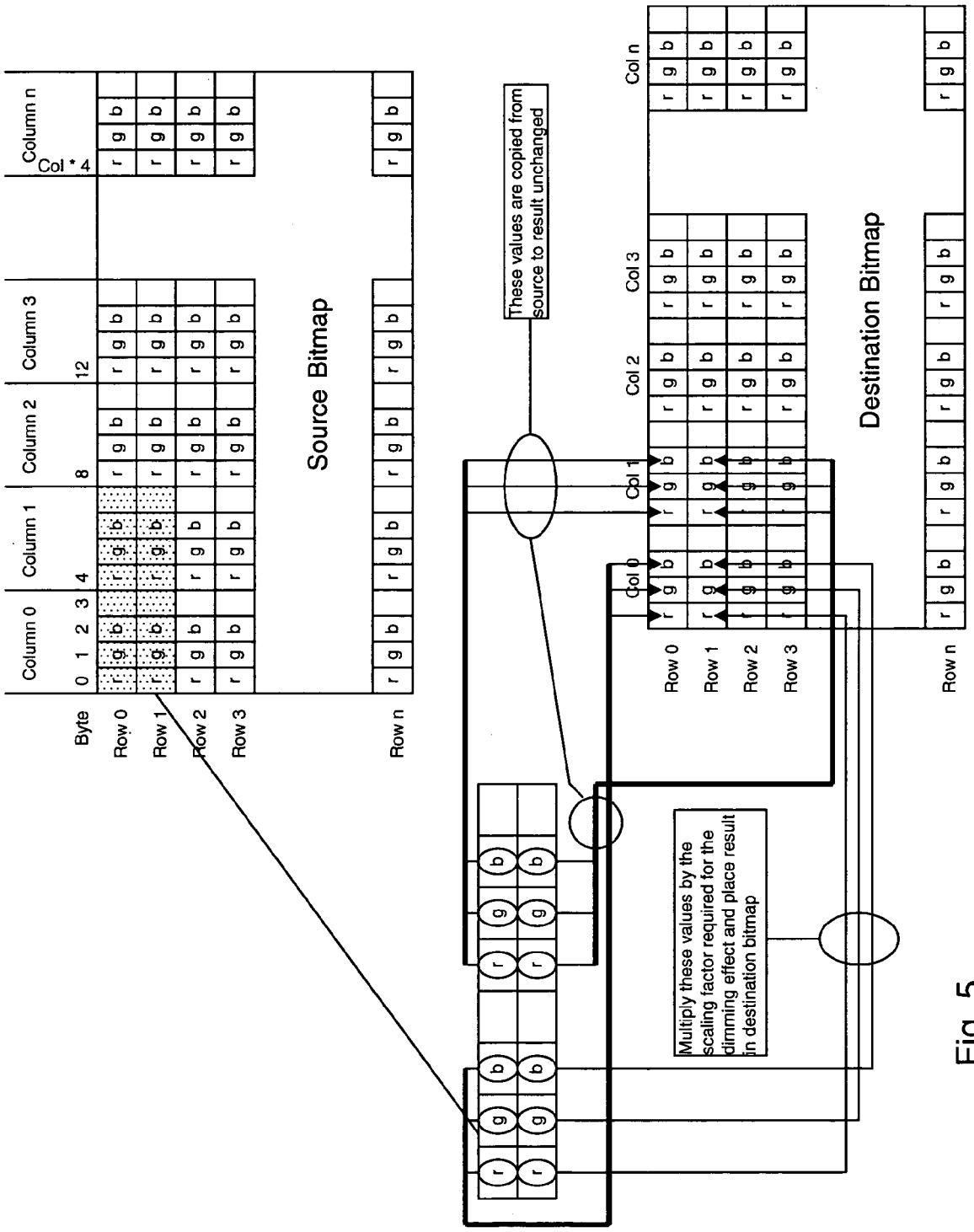
Source Bitmap
Destination Bitmap
These values are copied from source to result unchanged
Multiply these values by the scaling factor required for the dimming effect and place result in destination bitmap
Byte
Row 0
Row 1
Row 2
Row 3
Row n
Column 0
Column 1
Column 2
Column 3
Column n
Col * 4
Col 0
Col 1
Col 2
Col 3
Col n

Fig. 5

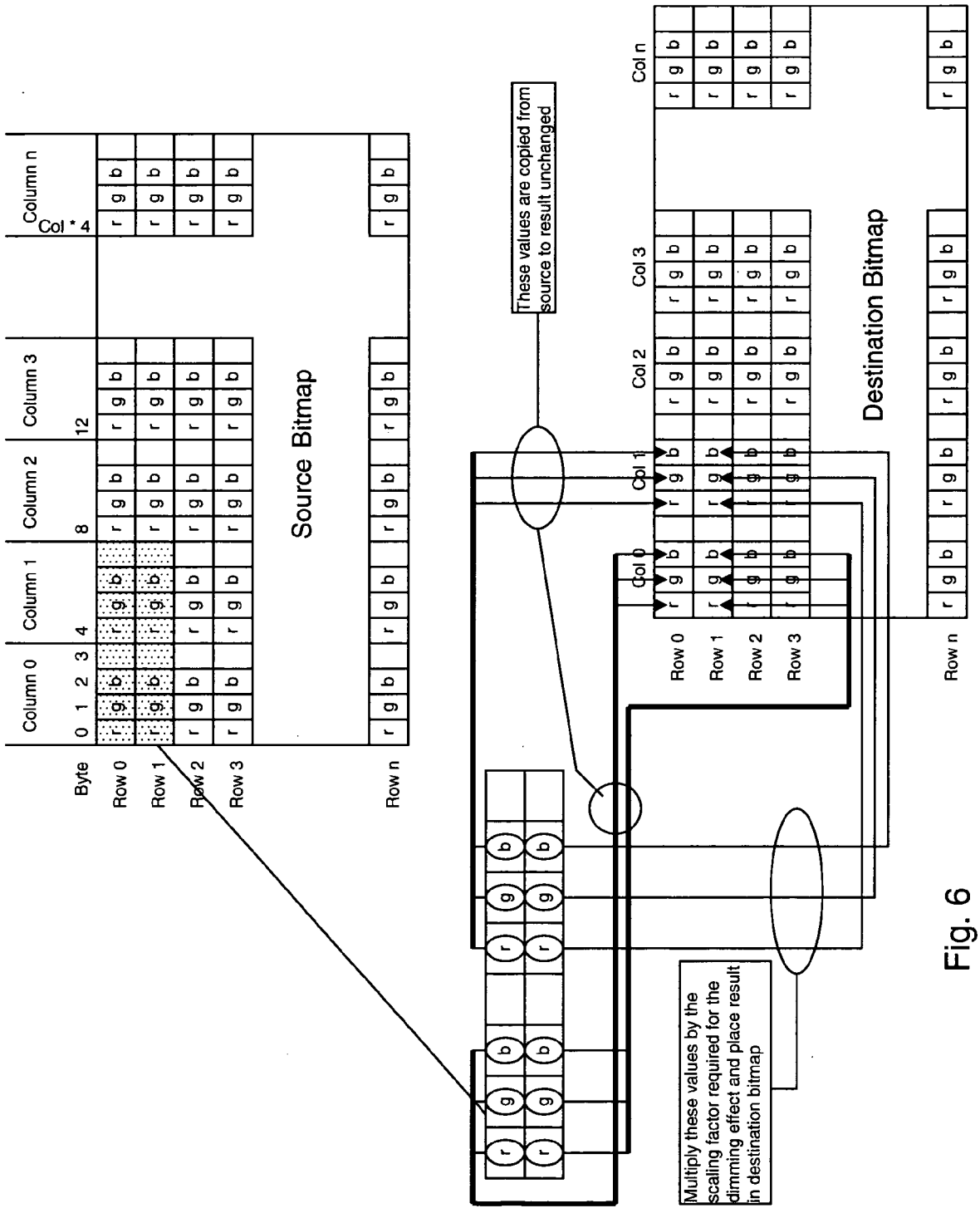
Source Bitmap
Destination Bitmap
These values are copied from source to result unchanged
Multiply these values by the scaling factor required for the dimming effect and place result in destination bitmap
Byte
Row 0
Row 1
Row 2
Row 3
Row n
Column 0
Column 1
Column 2
Column 3
Column n
Col * 4
Col 0
Col 1
Col 2
Col 3
Col n

Fig. 6

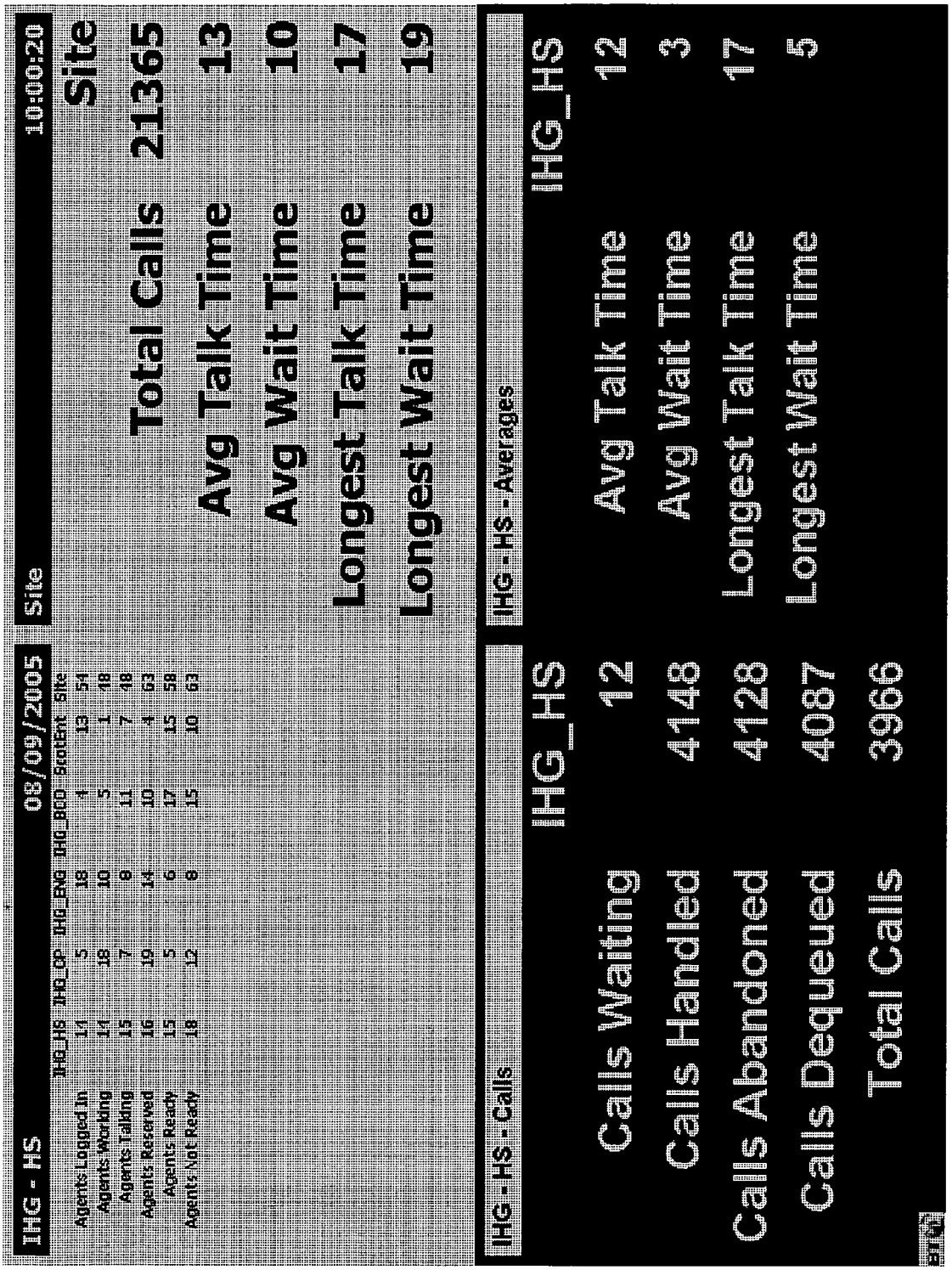
IHG - HS
08/09/2005
Site
10:00:20
IHG_HS IHG_OP IHG_BKG IHG_BOO StatEnt Site
Agents Logged In 14 5 18 4 13 54
Agents Working 14 18 10 5 1 48
Agents Talking 15 8 11 7 7 48
Agents Reserved 16 19 14 10 4 63
Agents Ready 15 5 6 17 15 58
Agents Not Ready 18 12 8 15 10 63
Site
Total Calls 21365
Avg Talk Time 13
Avg Wait Time 10
Longest Talk Time 17
Longest Wait Time 19
IHG - HS - Calls
IHG_HS
Calls Waiting 12
Calls Handled 4148
Calls Abandoned 4128
Calls Dequeued 4087
Total Calls 3966
IHG - HS - Averages
IHG_HS
Avg Talk Time 12
Avg Wait Time 3
Longest Talk Time 17
Longest Wait Time 5

Fig. 7

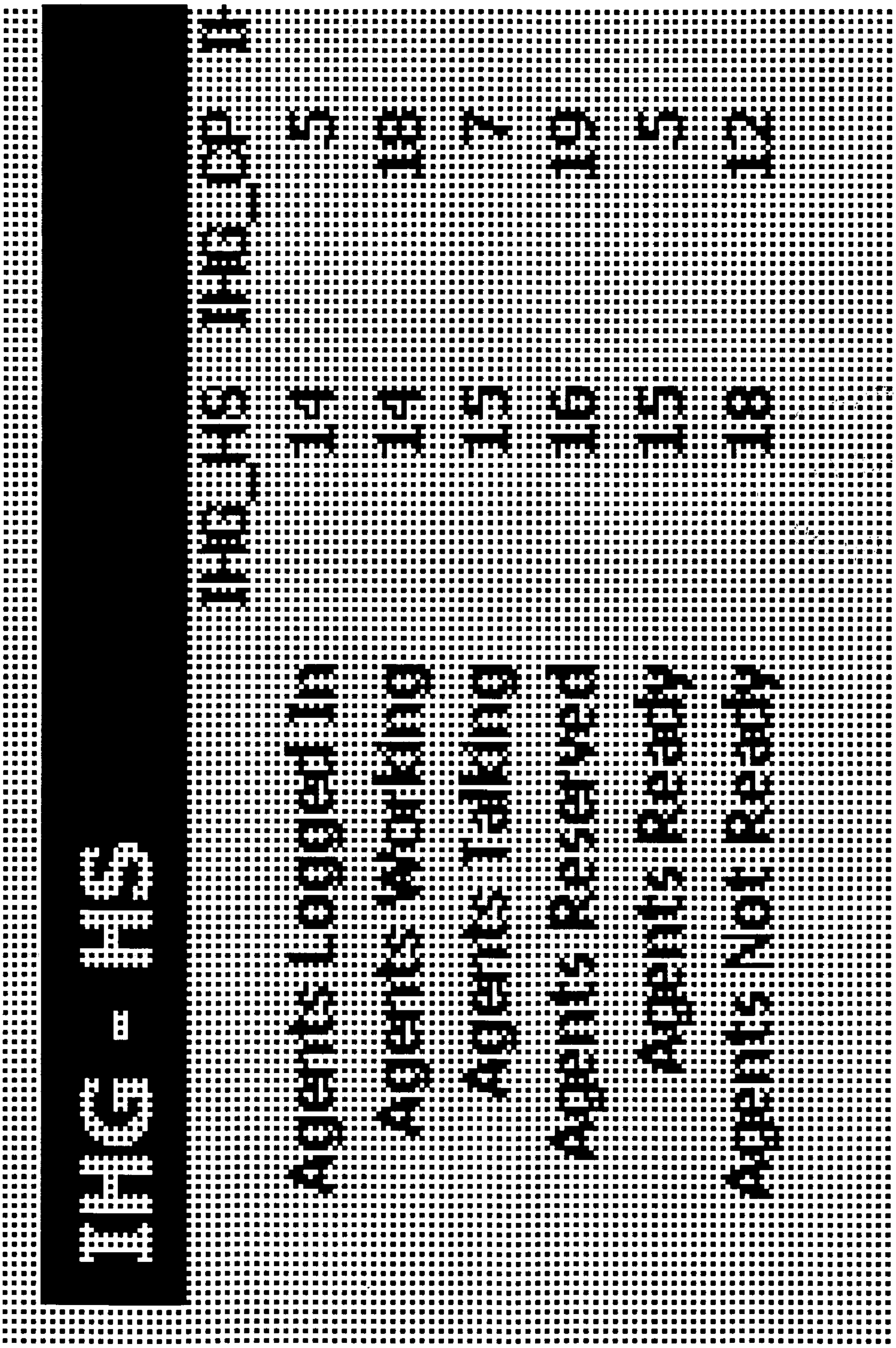
IHG - HS
IHG_HS
IHG_CP_D
Agents Logged In
14
5
Agents Working
14
18
Agents Talking
15
7
Agents Reserved
16
19
Agents Ready
15
5
Agents Not Ready
18
12

Fig. 8

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 05 25 6266

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (IPC)
A	US 2003/071769 A1 (SULLIVAN DAN ET AL) 17 April 2003 (2003-04-17) * paragraphs [0031], [0032], [0043]; figure 3 *	1,7,11, 12,18	G09G5/00
A	EP 0 965 974 A (PIONEER ELECTRONIC CORPORATION) 22 December 1999 (1999-12-22) * paragraphs [0045], [0053], [0054], [0057]; claims *	1,6,11, 12,17	
A	US 4 722 005 A (LEDENBACH ET AL) 26 January 1988 (1988-01-26) * abstract *	1,11,12	

TECHNICAL FIELDS SEARCHED (IPC)

G09G

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
The Hague	20 February 2006	Verhoof, P

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 05 25 6266

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

20-02-2006

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2003071769 A1	17-04-2003	WO 03034718 A1	24-04-2003
EP 0965974 A	22-12-1999	CN 1243301 A	02-02-2000
		JP 2000010522 A	14-01-2000
		US 2002167469 A1	14-11-2002
US 4722005 A	26-01-1988	JP 1112285 A	28-04-1989

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82

REFERENCES CITED IN THE DESCRIPTION

This list of references cited by the applicant is for the reader's convenience only. It does not form part of the European patent document. Even though great care has been taken in compiling the references, errors or omissions cannot be excluded and the EPO disclaims all liability in this regard.

Patent documents cited in the description

- US 6313878 B **[0014]**
- US 20030071769 A **[0014]**