

EUROPEAN PATENT APPLICATION

Application number: 88115124.5

Int. Cl.4: **G09G 1/28** , **H04N 5/262** ,
H04N 9/74

Date of filing: 15.09.88

A request for correction of the description has been filed pursuant to Rule 88 EPC. A decision on the request will be taken during the proceedings before the Examining Division (Guidelines for Examination in the EPO, A-V, 2.2).

Priority: 26.10.87 US 113030

Date of publication of application:
03.05.89 Bulletin 89/18

Designated Contracting States:
DE FR GB NL

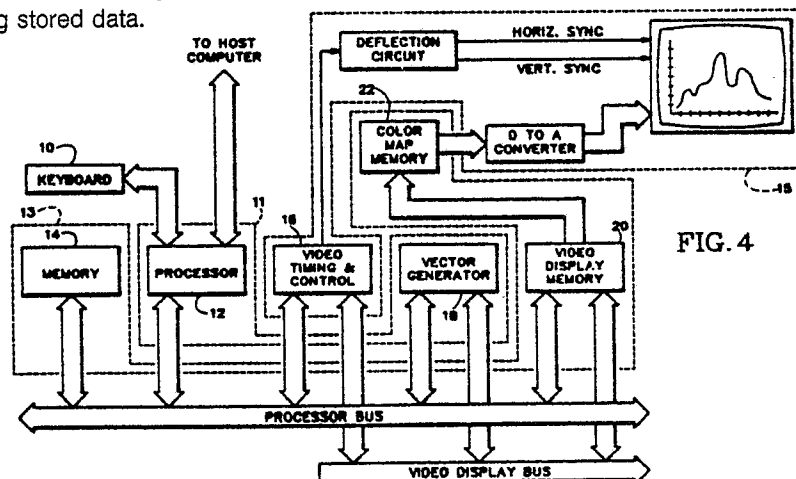
Applicant: **TEKTRONIX, INC.**
Howard Vollum Park 14150 Karl Braun Drive
P.O. Box 500
Beaverton Oregon 97224(US)

Inventor: **Dalrymple, John Charles**
7108 S.W. 12th Avenue
Portland Oregon 97219(US)
Inventor: **Bigger, Scott William**
8890 SW Scheckla
Tigard Oregon 97224(US)

Representative: **Liesegang, Roland, Dr.**
FORRESTER & BOEHMERT
Widenmayerstrasse 4
D-8000 München 22(DE)

Method and apparatus for representing three-dimensional color data in a one-dimensional reference system.

A method and apparatus are disclosed for representing and displaying the true (actual) color of an image in an index color system. The user specifies the number of color levels necessary to represent the range of displayable colors, and specifies the color (the red, green, blue primary color values) desired for an image. From the specified number, the apparatus calculates RGB (red, green, blue) color values, and stores them in memory in a manner suitable for referencing by an index type address. From the specified color, the apparatus calculates an index suitable for referencing the desired RGB color values from the stored color values. The apparatus comprises a keyboard and a port to a host computer for inputting index color data and true color data, a processing means coupled to the keyboard and/or host computer for producing index-type data from true color data, a memory means for storing the input data and data produced by the processing means, and a display means for displaying stored data.



METHOD AND APPARATUS FOR REPRESENTING THREE-DIMENSIONAL COLOR DATA IN A ONE-DIMENSIONAL REFERENCE SYSTEM

BACKGROUND OF THE INVENTION

5 This invention relates generally to the display of color images on a computer terminal and, more particularly, to a method of processing, storing, and referencing data representing image color.

In many computer terminal systems, image color data are stored as R,G,B (red, green, blue) color component values in predetermined memory locations. Often, red values are stored in one group of memory locations, green values in another, and blue values in a third group. To select a color, made up of
 10 one color value from each of the groups, the user usually specifies a color number or index. The index, which represents a single (one-dimensional) address, permits access to the particular set of RGB values representing the selected color. One such prior art "single index" system is described, for example, in U.S. Patent 4,509,043 issued April 2, 1985 to inventor P.X. Mossaides.

Another type of prior art system is the "true color" system, in which the true or actual colors of an object are sought to be displayed. One such true color system is described, for example, in the Model
 15 One/25 Programming Guide published December 12, 1983 by Raster Technologies Corporation. In such a true color system, the user generally specifies the RGB color values themselves (rather than color numbers or indexes) to represent a selected color. The "color values" approach is representative of the true color system--a three-dimensional color referencing system. The "color number" approach is representative of
 20 the index color system--a one-dimensional color referencing system.

True color systems are generally used to produce shaded images, to display scanned-in images, and to simulate physical phenomena such as colored lights illuminating a scene composed of colored objects. Indexed color systems are generally used to depict color of icon symbols, to represent scalar values, or to distinguish image segments (e.g., primitives such as points, lines, arcs, circles, rectangles, polygons, and
 25 text).

Often it is desirable to display symbolic icons, specified in index color, superimposed on shaded images specified in true color. To accomplish this, what is needed is a system which would permit both true-color representation and index color representation.

SUMMARY OF INVENTION

In accordance with the illustrated preferred embodiment of the invention, a method and apparatus is disclosed which provides for the display of both true-color images and index color images. The user may
 35 specify either a color number to denote index color, or RGB color values to denote true color. Index color operations and true color operations are performed in an index color environment. To perform true color operation in an index color environment, the apparatus computes one-dimensional index-type data from user-supplied three-dimensional true color data. The user specifies the number of color levels to be used in
 40 representing the range of displayable red, green and blue colors, and specifies the RGB values of the color to be displayed. The apparatus derives true color data (RGB color level values) from the number of color levels specified by the user, stores the derived data in memory in a predetermined arrangement suitable for referencing by a single address or index, and derives from the user-specified RGB values a single address for referencing the stored data.

The apparatus comprises a keyboard and a port to a host computer for inputting index color data and true color data, a processing means coupled to the keyboard and/or host computer for producing index-type data from true color data, a memory means for storing the input data and data produced by the processing means, and a display means for displaying stored data.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is block diagram of a system of the prior art capable of performing index color operations.

Fig. 2 is a block diagram showing an arrangement of data in a memory of the system of Fig. 1;

Fig. 3 is a block diagram of a system of the prior art capable of performing true color operations;

Fig. 4 is a block diagram of the apparatus of the present invention for performing index color operations and true color operations in an index color environment;

5 Fig. 5 is a block diagram showing a color map memory of the apparatus of Fig. 4;

Fig. 6 is a block diagram showing the range of displayable colors as a three-dimensional color coordinate space;

Fig. 7 is a block diagram of a concatenation mask used in the apparatus of Fig. 4; and

Fig. 8 is a block diagram showing multiple bit planes, a true color surface and a single index surface.

10

DESCRIPTION OF THE PREFERRED EMBODIMENT

15 Fig. 1 shows a system of the prior art capable of performing index color operations. Fig. 2 shows a color map memory of the system, RGB data typically stored in the memory, and a one-dimensional (single color index) address scheme for referencing stored data. Fig. 3 shows a three-dimensional address scheme of the prior art for referencing stored data. From user-supplied RGB values stored in image memory, references are made to color values stored in three look-up tables (LUTs). Fig. 2 shows a one address (index) scheme, while Fig. 3 shows a three address (true color) scheme.

20 Fig. 4 shows a true-color one-address scheme, the scheme utilized by the apparatus of the present invention. Like the prior art system described in U.S. patent 4,509,043, the apparatus of the present invention is capable of performing index color operations. Unlike prior art systems, the apparatus is capable of performing true color operations in an index color environment.

25 The apparatus comprises a keyboard 10 and port to a host computer for inputting index color data and true color data, a processing means 11 coupled to the keyboard and/or host computer for producing index-type data from true color data, a memory means 13 for storing the input data and data produced by the processing means, and a display means 15 for displaying stored data.

30 The processing means 11 includes a microprocessor 12 having a ROM (read only memory, not shown) with a stored program, and a vector generator 18. The memory means 13 includes a memory 14 for storing index information, a video display memory 20, and a color map memory 22. The display means 15 includes a video timing and control circuit 16, and a CRT (cathode ray tube) display with associated deflection circuit and D to A (digital-to-analog) converter.

35 The apparatus of the present invention is much like the system described in U.S. patent 4,509,043 for performing index operations, and the description of that system applies as well to the present apparatus. However, in the performing of true color operations in and color map memory of the present apparatus are designed to operate differently from said system, as described below.

40 Fig. 5 shows color map memory 22 of the present apparatus in greater detail. Like the prior art, the RGB values are arranged in tabular form in memory, with each memory location containing one set of R,G, and B values addressable by a single address. Unlike the prior art, the RGB values in the tables are calculated (derived) by the apparatus from data supplied to the apparatus. The calculation is performed by processor 12 under control of the stored program. (Examples of the stored program are presented in Appendixes A and B.) Also unlike the prior art, the single address used to reference the RGB values in the table is derived by processor 12 of the apparatus from RGB values specified by the user. Thus, in the present invention, both the RGB table values and the single address for referencing the RGB table values are derived by the apparatus.

45 The initial user of the apparatus (e.g., the supplier of the apparatus) specifies the RGB color values to be used in populating the table, by uniformly quantizing the closed interval from zero to one for each color. For example, the user may specify the number of the displayable levels to be Q (i.e., Q discrete quantization levels of red, green, and blue displayable primary colors). Thereafter, the apparatus calculates the actual table entries (i.e., the true-color RGB values which, when referenced subsequently by a single index-type address, produces the original user-specified values).

As shown in Table I, one set of displayable primary color levels may be:

55

TABLE I

5 10 15 20 25	Number (Q) of Quantization Levels = 5	Quantization Level	Displayable Primary Color Levels		
			R	G	B
		0	0	0	0
		1	.25	.25	.25
		2	.5	.5	.5
		3	.75	.75	.75
		4	1	1	1

The five quantization levels occupy the range from 0% to 100% of displayable red, green, and blue primary colors.

As shown in Fig. 6, the range of displayable colors may be viewed as a three-dimensional color space, where (0,0,0) represents black, i.e., 0% red, 0% green, 0% blue, (1,0,0) represents 100% red, (0,1,0) represents 100% green, (0,0,1) represents 100% blue, and (1,1,1) represents white, i.e., 100% red, 100% green and 100% blue.

As shown in Fig. 6, the user of the apparatus may select:

- Q₁ levels of Red
- Q₂ levels of Green
- Q₃ levels of Blue

when representing a true-color object within the limits of the displayable colors of the apparatus. From the displayable levels specified by the user, the apparatus derives (calculates) color values suitable for single index referencing (single addressing), and stores the derived values in tables in memory for later use.

The apparatus calculates the color values (table entries) and populates the RGB tables as follows:

Red (address) = $i/(Q_1 - 1)$
 Green (address) = $j/(Q_2 - 1)$
 Blue (address) = $k/(Q_3 - 1)$

where:

Q₁ represents the number of quantization levels specified for red
 Q₂ represents the number of quantization levels specified for green
 Q₃ represents the number of quantization levels specified for blue
 quantization level $i = 0$ to $Q_1 - 1$
 quantization level $j = 0$ to $Q_2 - 1$
 quantization level $k = 0$ to $Q_3 - 1$
 address = $i + Q_1 \cdot j + Q_1 \cdot Q_2 \cdot k$
 n represents the number of permissible table entries

and

$$(Q_1 \times Q_2 \times Q_3) < n.$$

or

$$Q_1 \times Q_2 \times Q_3 \leq 2^m, \text{ where } m \text{ represents the number of bit planes, as shown for example in Figs. 2 and 8.}$$

5 The following RGB tables show color data produced (derived) by processor 12 in response to user-specified quantization levels. Processor 12 stores the derived data in memory 22 in a predetermined sequence (arrangement), as shown in Table II below.

10 **TABLE II**

15	<u>Specified Color Values</u>	<u>Derived Single Address</u>	<u>Derived & Stored Color Data</u>		
20			R	G	B
	(0,0,0)	0	0	0	0
25	(.25,0,0)	1	.25	0	0
	(.5,0,0)	2	.5	0	0
30			.75	0	0
			1	0	0
35	•	•	0	.25	0
40			.25	.25	0
			.5	.25	0
45			.75	.25	0
50			1	.25	0
	•	•	0	.5	0
55			.25	.5	0

5

10

15

20

25

30

35

40

45

50

55

(.5,1,0) 22**(.75,1,0) 23****(1,1,0) 24****(0,0,.25) 25****(.25,0,.25) 26**

.5	.5	0
.75	.5	0
1	.5	0
0	.75	0
.25	.75	0
.5	.75	0
.75	.75	0
1	.75	0
0	1	0
.25	1	0
.5	1	0
.75	1	0
1	1	0
0	0	.25
.25	0	.25
.5	0	.25
.75	0	.25

5

10

15

20

25

30

35

40

45

50

55

1	0	.25
0	.25	.25
.25	.25	.25
.5	.25	.25
.75	.25	.25
1	.25	.25
0	.5	.25
.25	.5	.25
.5	.5	.25
.75	.5	.25
1	.5	.25
0	.75	.25
.25	.75	.25
.5	.75	.25
.75	.75	.25
1	.75	.25
0	1	.25

5

10

15

20

25

30

35

40

45

50

55

(.75,1,25) 48

(1,1,.25) 49

(0,0,5) 50

(.25,0,.5) 51

•

•

•

•

.25	1	.25
.5	1	.25
.75	1	.25
1	1	.25
0	0	.5
.25	0	.5
.5	0	.5
.75	0	.5
1	0	.5
0	.25	.5
.25	.25	.5
.5	.25	.5
.75	.25	.5
1	.25	.5
0	.5	.5
.25	.5	.5
.5	.5	.5

5

10

15

20

25

30

35

40

45

50

55

(.75,1,.5)**73****(1,1,.5)****74****(0,0,.75)****75****(.25,0,.75)****76**

.75	.5	.5
1	.5	.5
0	.75	.5
.25	.75	.5
.5	.75	.5
.75	.75	.5
1	.75	.5
0	1	.5
.25	1	.5
.5	1	.5
.75	1	.5
1	1	.5
0	0	.75
.25	0	.75
.5	0	.75

5

10

15

20

25

30

35

40

45

50

55

(.25,.5,.75) 86

.75	0	.75
1	0	.75
0	.25	.75
.25	.25	.75
.5	.25	.75
.75	.25	.75
1	.25	.75
0	.5	.75
.25	.5	.75
.5	.5	.75
.75	.5	.75
1	.5	.75
0	.75	.75
.25	.75	.75
.5	.75	.75
.75	.75	.75

5

10

(.75,1,.75) 98

15

(1,1,.75) 99

(0,0,1) 100

20

(.25,0,1) 101

25

30

• •

35

40

• •

45

50

55

1	.75	.75
0	1	.75
.25	1	.75
.5	1	.75
.75	1	.75
1	1	.75
0	0	1
.25	0	1
.5	0.5	1
.75	.0	1
1	0	1
0	.25	1
.25	.25	1
.5	.25	1
.75	.25	1
1	.25	1

			0	.5	1
5	•	•	.25	.5	1
			.5	.5	1
10			.75	.5	1
			1	.5	1
15			0	.75	1
			.25	.75	1
20			.5	.75	1
			.75	.75	1
25	•	•	1	.75	1
			0	1	1
30			.25	1	1
			.5	1	1
35	•	•	.75	1	1
			1	1	1
40					
	(.75, 1, 1)	123			
45					
	(1, 1, 1)	124			

The sequence in which the data is stored in memory 22 ensures that, from the RGB values specified by the user, the processor 12 will derive the appropriate index (single address) for referencing the correct RGB values in memory 22. Processor 12 derives each single address (index) by means of the following formula:

$$\text{Index} = [R (Q_1 - 1)] \text{ rounded} + Q_1 [G (Q_2 - 1)] \text{ rounded} + Q_1 Q_2 [B (Q_2 - 1)] \text{ rounded}.$$

Thus, as shown in Table I above, from the three-dimensional true color value (.25, .5, .75) specified by the user, the address 86 would be derived by processor 12, permitting access to the correct RGB values in the table.

An alternative formula for deriving single addresses (indexes) from specified (input) RGB values is:

$$[RQ_1]_{\text{truncated}} + Q_1 [G Q_2]_{\text{truncated}} + Q_1 Q_2 [B Q_3]_{\text{truncated}}$$

where: $0 \leq R < 1$; $0 \leq G < 1$; $0 \leq B < 1$

5 and Q_1, Q_2, Q_3 , are powers of two.

Given B_1 bits of red, B_2 bits of green and B_3 bits of blue, and given an m -bit plane surface, as shown for example in Fig. 8, where $m = b_1 + b_2 + b_3$ and $b_1 \leq B_1, b_2 \leq B_2, b_3 \leq B_3$, then the above-mentioned truncation operation may be performed as shown in Fig. 7, where the b_1 most significant bits (MSB) of the R field, and the b_2 MSB of the G field, and the b_3 MSB of the B field, are concatenated, and shifted to align
 10 it with the m -bit

In the foregoing, the method for converting from true color data to single index data is described. The method include the steps of deriving true color data (i.e., RGB color level values) from the number of color levels specified by the user, storing the derived data in memory in predetermined arrangement suitable for referencing by a single address or index, and deriving from user-supplier RGB color values a single
 15 address for referencing the stored data.

Appendix A shows RGB values, and associated single index addresses, derived for:

$Q_1 = 5$ red levels

$Q_2 = 5$ green levels

$Q_3 = 5$ blue levels,

20 the RGB values being in the range 0 to 255, representing 0% to 100%.

Appendix B shows RGB values, and associated single index addresses, derived for:

$Q_1 = 7$ red levels

$Q_2 = 6$ green levels

$Q_3 = 5$ blue levels,

25 the RGB values being in the range 0 to 255, representing 0% to 100%.

In appendix C below, index and true color data (stored in binary form in color map memory 22) are shown for both a single index surface (surface 1) and a true color surface (surface 2). For example, of the address "00 01 10 11" shown in Appendix C and in Fig. 8, the "00" represents a color index for surface 1, and the "01 10 11" represents RGB color values for surface 2. The address was computed (derived) from
 30 user-supplied index and true color values. The address is used to reference the RGB values "01010101 10101010 11111111".

Depending on which surface has priority over the other, the RGB values may differ for a given address. For example, given the computed address "01 11 11 11", if the RGB values "10000000 10000000 10000000" had been predetermined to represent the color index "01", and the RGB values "11111111 11111111 11111111" had been predetermined to represent the RGB color level "11 11 11", then the RGB
 35 values "10000000 10000000 10000000" would be used if the indexed color surface had priority, and the RGB values "11111111 11111111 11111111" would be used if the true color surface had priority.

40

45

50

55

APPENDIX A
(JGRAM LISTING/DATA/COMMENTS)

```

5
10
15
20
25
30
35
40
45
50
55

/* global type and data declarations */

typedef struct {
    unsigned char R, G, B;
} table_entry_t; /* "unsigned char" data is 8 bits in length */

/* In the invention, variables q1, q2, and q3 are set in
   in response to a user command or set to default values
   by the system. */

int q1, /* number of quanta of red */
int q2, /* number of quanta of green */
int q3; /* number of quanta of blue */

#include "tc_utils.c" /* the "populate" and "convert" functions */

/*****

/* The following test program calls the "populate" and "convert"
   functions and prints their results for some example cases */

main()
{
    table_entry_t array[5*5*5];
    int i;

    q1 = 5; /* number of red quantum levels */
    q2 = 5; /* number of green quantum levels */
    q3 = 5; /* number of blue quantum levels */

    populate(array);

    for(i=0; i <= 5*5*5-1; ++i)
        printf ("%d:\t%d\t%d\t%d\n", i,
                (int)array[i].R, (int)array[i].G, (int)array[i].B),

        printf("%d\n", convert(0.0, 0.0, 0.0));
        printf("%d\n", convert(0.0, 0.0, 0.25));
        printf("%d\n", convert(0.0, 0.0, 0.5));
        printf("%d\n", convert(0.0, 0.0, 0.75));
        printf("%d\n", convert(0.0, 0.0, 1.0));
        printf("%d\n", convert(0.0, 0.25, 0.0));
        printf("%d\n", convert(0.0, 0.5, 0.0));
        printf("%d\n", convert(0.0, 0.75, 0.0));
        printf("%d\n", convert(0.0, 1.0, 0.0));
        printf("%d\n", convert(0.25, 0.0, 0.0));
        printf("%d\n", convert(0.5, 0.0, 0.0));
        printf("%d\n", convert(0.75, 0.0, 0.0));
        printf("%d\n", convert(1.0, 0.0, 0.0));

```

```

5  /*****
   /* populate.c -- example color-map population function. This function
   computes "surface color-map" (per Mossaides) data for simulating true
   color, and stores the data in a main-memory array of RGB values. The
   location of the array is passed into this function as an argument.
10  This function is called in response to a user command to create a
   "true-color surface."

   After this function has been called, the array can be copied into
   the hardware color map, or combined with other "surface color maps" as
   per Mossaides, and the combined map copied into the hardware color
   map. The hardware color map is assumed to hold 8 bits each for red,
15  green, and blue entries at each address.

   Note that the computation of the address shown below is only one
   example; another equally valid example would have address = i*q2*q3
   + j*q3 + k; and others are possible. The only requirement is that the
   same rule be used to compute the address in this function and to compute
   the index in the "convert" function. */
20  populate(table)
      table_entry_t *table; /* pointer to color table array
                              in main memory */
{
    int i; /* red quantum number */
    int j; /* green quantum number */
25  int k; /* blue quantum number */
    int address;
    double red_value, green_value, blue_value;

    /* loop for all possible sets of quanta */
    for (i=0; i <= q1-1; ++i) /* all the red quanta */
        for (j=0; j <= q2-1; ++j) /* all the green quanta */
30  for (k=0; k <= q3-1; ++k) /* all the blue quanta */

        /* do the following for each set of quanta */
        {
            /* compute an array address corresponding to
               the current set of quanta */
35  address = i + q1*j + q1*q2*k;

            /* compute the "ideal" primary levels from the
               quantum numbers and the maximum quantum numbers */

            red_value = (double)i/(double)(q1-1);
            green_value = (double)j/(double)(q2-1);
40  blue_value = (double)k/(double)(q3-1);

            /* convert the "ideal" primary levels (real numbers
               from 0.0 to 1.0) to "actual" primary levels (unsigned
               chars ranging from 0 to 255) and store them in the array,
               by scaling and rounding */
45
50
55

```

```

5         table[address].R = (unsigned char)(255 * red_value + 0.5);
          table[address].G = (unsigned char)(255 * green_value + 0.5);
          table[address].B = (unsigned char)(255 * blue_value + 0.5);

          } /* end of loop body */
    }

    /*****
10  /* convert.c -- this function takes as input three real numbers R, G,
     and B (on the range 0.0 to 1.0) representing the color of a pixel, and
     produces a single color index to be stored at that pixel's location in
     the frame buffer. This function is used during the course of drawing
     image segments into the frame buffer. */

15  convert(r, g, b)
     double r, g, b;
    {
        int i, j, k;

        /* compute quantum numbers of nearest available RGB values. */
20        i = (int)(r * (q1-1) + 0.5); /* red quantum number */
        j = (int)(g * (q2-1) + 0.5); /* green quantum number */
        k = (int)(b * (q3-1) + 0.5); /* blue quantum number */

        /* Combine the quantum numbers into an index according to
           the same rule as is used for computing the address variable
           in the "populate" function. Return the index as a value
25         to the caller of the "convert" function.

           The caller can subsequently write the computed index into
           the frame buffer, when it wants to simulate writing color (r,g,b)
           at one or more pixel locations the frame buffer. */

        return (i + q1*j + q1*q2*k);
30    }

```

35

40

45

50

55

EP 0 313 789 A1

	0:	0	0	0
	1:	64	0	0
	2:	128	0	0
5	3:	191	0	0
	4:	255	0	0
	5:	0	64	0
	6:	64	64	0
	7:	128	64	0
	8:	191	64	0
10	9:	255	64	0
	10:	0	128	0
	11:	64	128	0
	12:	128	128	0
	13:	191	128	0
15	14:	255	128	0
	15:	0	191	0
	16:	64	191	0
	17:	128	191	0
	18:	191	191	0
20	19:	255	191	0
	20:	0	255	0
	21:	64	255	0
	22:	128	255	0
	23:	191	255	0
25	24:	255	255	0
	25:	0	0	64
	26:	64	0	64
	27:	128	0	64
	28:	191	0	64
30	29:	255	0	64
	30:	0	64	64
	31:	64	64	64
	32:	128	64	64
	33:	191	64	64
35	34:	255	64	64
	35:	0	128	64
	36:	64	128	64
	37:	128	128	64
	38:	191	128	64
40	39:	255	128	64
	40:	0	191	64
	41:	64	191	64
	42:	128	191	64
	43:	191	191	64
45	44:	255	191	64
	45:	0	255	64
	46:	64	255	64
	47:	128	255	64
	48:	191	255	64
50	49:	255	255	64
	50:	0	0	128
	51:	64	0	128
	52:	128	0	128
	53:	191	0	128
55	54:	255	0	128
	55:	0	64	128

	56:	64	64	128
	57:	128	64	128
	58:	191	64	128
5	59:	255	64	128
	60:	0	128	128
	61:	64	128	128
	62:	128	128	128
	63:	191	128	128
10	64:	255	128	128
	65:	0	191	128
	66:	64	191	128
	67:	128	191	128
	68:	191	191	128
	69:	255	191	128
15	70:	0	255	128
	71:	64	255	128
	72:	128	255	128
	73:	191	255	128
	74:	255	255	128
20	75:	0	0	191
	76:	64	0	191
	77:	128	0	191
	78:	191	0	191
	79:	255	0	191
25	80:	0	64	191
	81:	64	64	191
	82:	128	64	191
	83:	191	64	191
	84:	255	64	191
30	85:	0	128	191
	86:	64	128	191
	87:	128	128	191
	88:	191	128	191
	89:	255	128	191
35	90:	0	191	191
	91:	64	191	191
	92:	128	191	191
	93:	191	191	191
	94:	255	191	191
40	95:	0	255	191
	96:	64	255	191
	97:	128	255	191
	98:	191	255	191
	99:	255	255	191
45	100:	0	0	255
	101:	64	0	255
	102:	128	0	255
	103:	191	0	255
	104:	255	0	255
50	105:	0	64	255
	106:	64	64	255
	107:	128	64	255
	108:	191	64	255
	109:	255	64	255
55	110:	0	128	255
	111:	64	128	255

	112:	128	128	255
	113:	191	128	255
5	114:	255	128	255
	115:	0	191	255
	116:	64	191	255
	117:	128	191	255
	118:	191	191	255
10	119:	255	191	255
	120:	0	255	255
	121:	64	255	255
	122:	128	255	255
	123:	191	255	255
15	124:	255	255	255
	0			
	25			
	50			
	75			
20	100			
	5			
	10			
	15			
	20			
25	1			
	2			
	3			
	4			
30				
35				
40				
45				
50				
55				

APPENDIX B

(Program Listing/Data/Comments)

```

5
/* global type and data declarations */

10 typedef struct {
    unsigned char R, G, B;
} table_entry_t; /* "unsigned char" data is 8 bits in length */

/* In the invention, variables q1, q2, and q3 are set in
   in response to a user command or set to default values
   by the system. */

15 int q1; /* number of quanta of red */
    int q2; /* number of quanta of green */
    int q3; /* number of quanta of blue */

#include "tc_utils.c" /* the "populate" and "convert" functions */

20 /* The following test program calls the "populate" and "convert"
   functions and prints their results for some example cases */

main()
{
25     table_entry_t array[7*6*5];
    int i;

    q1 = 7; /* number of red quantum levels */
    q2 = 6; /* number of green quantum levels */
    q3 = 5; /* number of blue quantum levels */

30     populate(array);

    for(i=0; i <= 7*6*5-1; ++i)
        printf ("%d:\t%d\t%d\t%d\n", i,
                (int)array[i].R, (int)array[i].G, (int)array[i].B);

    printf("%d\n", convert(0.0, 0.0, 0.0));
35     printf("%d\n", convert(0.0, 0.0, 0.25));
    printf("%d\n", convert(0.0, 0.0, 0.5));
    printf("%d\n", convert(0.0, 0.0, 0.75));
    printf("%d\n", convert(0.0, 0.0, 1.0));
    printf("%d\n", convert(0.0, 0.2, 0.0));
    printf("%d\n", convert(0.0, 0.4, 0.0));
    printf("%d\n", convert(0.0, 0.6, 0.0));
40     printf("%d\n", convert(0.0, 0.8, 0.0));
    printf("%d\n", convert(0.1666667, 0.0, 0.0));
    printf("%d\n", convert(0.3333333, 0.0, 0.0));
    printf("%d\n", convert(0.5000000, 0.0, 0.0));
    printf("%d\n", convert(0.6666667, 0.0, 0.0));
    printf("%d\n", convert(0.8333333, 0.0, 0.0));
45 }

```

50

55

```

5  /*****
   * populate.c -- example color-map population function. This function
   * computes "surface color-map" (per Mossaides) data for simulating true
   * color, and stores the data in a main-memory array of RGB values. The
   * location of the array is passed into this function as an argument.
10  This function is called in response to a user command to create a
   * "true-color surface."

   After this function has been called, the array can be copied into
   the hardware color map, or combined with other "surface color maps" as
   per Mossaides, and the combined map copied into the hardware color
15  map. The hardware color map is assumed to hold 8 bits each for red,
   green, and blue entries at each address.

   Note that the computation of the address shown below is only one
   example: another equally valid example would have address = i + q1*j +
   q1*q2*k and others are possible. The only requirement is that the same
   rule be used to compute the address in this function and to compute
20  the index in the "convert" function. */

populate(table)
    table_entry_t *table; /* pointer to color table array
                           in main memory */
{
    int i; /* red quantum number */
    int j; /* green quantum number */
25  int k; /* blue quantum number */
    int address;
    double red_value, green_value, blue_value;

    /* loop for all possible sets of quanta */
    for (i=0; i <= q1-1; ++i) /* all the red quanta */
30      for (j=0; j <= q2-1; ++j) /* all the green quanta */
          for (k=0; k <= q3-1; ++k) /* all the blue quanta */

            /* do the following for each set of quanta */
            {
                /* compute an array address corresponding to
                   the current set of quanta */
35                address      = i*q2*q3 + j*q3 + k;

                /* compute the "ideal" primary levels from the
                   quantum numbers and the maximum quantum numbers */

                red_value     = (double)i/(double)(q1-1);
20                green_value = (double)j/(double)(q2-1);
                blue_value    = (double)k/(double)(q3-1);

                /* convert the "ideal" primary levels (real numbers
                   from 0.0 to 1.0) to "actual" primary levels (unsigned
                   chars ranging from 0 to 255) and store them in the array,
                   by scaling and rounding */
45
            }
}
50
55

```

```

5         table[address].R = (unsigned char)(255 * red_value + 0.5);
          table[address].G = (unsigned char)(255 * green_value + 0.5);
          table[address].B = (unsigned char)(255 * blue_value + 0.5);

          } /* end of loop body */
    }

    /*****

10    /* convert c -- this function takes as input three real numbers R, G,
       and B (on the range 0.0 to 1.0) representing the color of a pixel, and
       produces a single color index to be stored at that pixel's location in
       the frame buffer. This function is used during the course of drawing
       image segments into the frame buffer. */

15    convert(r, g, b)
        double r, g, b;
    {
        int i, j, k;

        /* compute quantum numbers of nearest available RGB values */
        i = (int)(r * (q1-1) + 0.5); /* red quantum number */
20        j = (int)(g * (q2-1) + 0.5); /* green quantum number */
        k = (int)(b * (q3-1) + 0.5); /* blue quantum number */

        /* Combine the quantum numbers into an index according to
           the same rule as is used for computing the address variable
           in the "populate" function. Return the index as a value
           to the caller of the "convert" function.

25        The caller can subsequently write the computed index into
           the frame buffer, when it wants to simulate writing color (r,g,b)
           at one or more pixel locations the frame buffer. */

        return (i*q2*q3 + j*q3 + k);
    }
30

```

35

40

45

50

55

	0:	0	0	0
	1:	0	0	64
	2:	0	0	128
5	3:	0	0	191
	4:	0	0	255
	5:	0	51	0
	6:	0	51	64
	7:	0	51	128
10	8:	0	51	191
	9:	0	51	255
	10:	0	102	0
	11:	0	102	64
	12:	0	102	128
15	13:	0	102	191
	14:	0	102	255
	15:	0	153	0
	16:	0	153	64
	17:	0	153	128
20	18:	0	153	191
	19:	0	153	255
	20:	0	204	0
	21:	0	204	64
	22:	0	204	128
25	23:	0	204	191
	24:	0	204	255
	25:	0	255	0
	26:	0	255	64
	27:	0	255	128
	28:	0	255	191
30	29:	0	255	255
	30:	43	0	0
	31:	43	0	64
	32:	43	0	128
	33:	43	0	191
35	34:	43	0	255
	35:	43	51	0
	36:	43	51	64
	37:	43	51	128
	38:	43	51	191
40	39:	43	51	255
	40:	43	102	0
	41:	43	102	64
	42:	43	102	128
	43:	43	102	191
45	44:	43	102	255
	45:	43	153	0
	46:	43	153	64
	47:	43	153	128
	48:	43	153	191
50	49:	43	153	255
	50:	43	204	0
	51:	43	204	64
	52:	43	204	128
	53:	43	204	191
55	54:	43	204	255
	55:	43	255	0

EP 0 313 789 A1

	56:	43	255	64
	57:	43	255	128
	58:	43	255	191
5	59:	43	255	255
	60:	85	0	0
	61:	85	0	64
	62:	85	0	128
	63:	85	0	191
10	64:	85	0	255
	65:	85	51	0
	66:	85	51	64
	67:	85	51	128
	68:	85	51	191
15	69:	85	51	255
	70:	85	102	0
	71:	85	102	64
	72:	85	102	128
	73:	85	102	191
20	74:	85	102	255
	75:	85	153	0
	76:	85	153	64
	77:	85	153	128
	78:	85	153	191
25	79:	85	153	255
	80:	85	204	0
	81:	85	204	64
	82:	85	204	128
	83:	85	204	191
30	84:	85	204	255
	85:	85	255	0
	86:	85	255	64
	87:	85	255	128
	88:	85	255	191
35	89:	85	255	255
	90:	128	0	0
	91:	128	0	64
	92:	128	0	128
	93:	128	0	191
40	94:	128	0	255
	95:	128	51	0
	96:	128	51	64
	97:	128	51	128
	98:	128	51	191
45	99:	128	51	255
	100:	128	102	0
	101:	128	102	64
	102:	128	102	128
	103:	128	102	191
	104:	128	102	255
50	105:	128	153	0
	106:	128	153	64
	107:	128	153	128
	108:	128	153	191
	109:	128	153	255
55	110:	128	204	0
	111:	128	204	64

	112:	128	204	128
	113:	128	204	191
	114:	128	204	255
5	115:	128	255	0
	116:	128	255	64
	117:	128	255	128
	118:	128	255	191
	119:	128	255	255
10	120:	170	0	0
	121:	170	0	64
	122:	170	0	128
	123:	170	0	191
	124:	170	0	255
15	125:	170	51	0
	126:	170	51	64
	127:	170	51	128
	128:	170	51	191
	129:	170	51	255
20	130:	170	102	0
	131:	170	102	64
	132:	170	102	128
	133:	170	102	191
	134:	170	102	255
25	135:	170	153	0
	136:	170	153	64
	137:	170	153	128
	138:	170	153	191
	139:	170	153	255
30	140:	170	204	0
	141:	170	204	64
	142:	170	204	128
	143:	170	204	191
	144:	170	204	255
35	145:	170	255	0
	146:	170	255	64
	147:	170	255	128
	148:	170	255	191
	149:	170	255	255
40	150:	213	0	0
	151:	213	0	64
	152:	213	0	128
	153:	213	0	191
	154:	213	0	255
45	155:	213	51	0
	156:	213	51	64
	157:	213	51	128
	158:	213	51	191
	159:	213	51	255
50	160:	213	102	0
	161:	213	102	64
	162:	213	102	128
	163:	213	102	191
	164:	213	102	255
55	165:	213	153	0
	166:	213	153	64
	167:	213	153	128

EP 0 313 789 A1

	168:	213	153	191
	169:	213	153	255
	170:	213	204	0
5	171:	213	204	64
	172:	213	204	128
	173:	213	204	191
	174:	213	204	255
	175:	213	255	0
10	176:	213	255	64
	177:	213	255	128
	178:	213	255	191
	179:	213	255	255
	180:	255	0	0
15	181:	255	0	64
	182:	255	0	128
	183:	255	0	191
	184:	255	0	255
	185:	255	51	0
20	186:	255	51	64
	187:	255	51	128
	188:	255	51	191
	189:	255	51	255
	190:	255	102	0
25	191:	255	102	64
	192:	255	102	128
	193:	255	102	191
	194:	255	102	255
	195:	255	153	0
30	196:	255	153	64
	197:	255	153	128
	198:	255	153	191
	199:	255	153	255
	200:	255	204	0
35	201:	255	204	64
	202:	255	204	128
	203:	255	204	191
	204:	255	204	255
	205:	255	255	0
	206:	255	255	64
40	207:	255	255	128
	208:	255	255	191
	209:	255	255	255
	0			
	1			
45	2			
	3			
	4			
	5			
	10			
50	15			
	20			
	30			
	60			
	90			
55	120			
	150			

5

APPENDIX C

{Program Listing/Data/Comments}

10

EXAMPLE SHOWING TRUE COLOR SURFACE AND INDEXED COLOR SURFACE
 John C. Dalrymple
 10/21/87

15 Assume that the indexed surface is on bit planes 7 and 6 and that the true-color surface is on bit planes 5 down to 0, with planes 5 and 4 for red, 3 and 2 for green, and 1 and 0 for blue.

Assume that there are 8 bits per primary (R, G, or B) in the data fields of the hardware color map, where the bits represent fractional values with 00000000 representing 0 and 11111111 representing unity.

20 Assume for the example that the indexed-surface color map has been set by the user to:

Surface index	R	G	B
00	(transparent)	(transparent)	(transparent)
01	10000000	10000000	10000000
10	11111111	11111111	00000000
11	00000000	10000000	11111111

Assume further that the user has set the background color (behind all surfaces) to R = 0, G = 0, B = 0 (black).

30 Case 1. User has assigned higher priority to the indexed color surface.

Then the full hardware color map would be populated as:

bit numbers
 76 54 32 10

35

color map address	R	G	B	(remarks)
00 00 00 00	00000000	00000000	00000000	(background color)
00 00 00 01	00000000	00000000	01010101	(from true-color surf.)
00 00 00 10	00000000	00000000	10101010	"
00 00 00 11	00000000	00000000	11111111	"
00 00 01 00	00000000	01010101	11111111	"
00 00 01 01	00000000	01010101	11111111	"
00 00 01 10	00000000	01010101	11111111	"
00 00 01 11	00000000	01010101	11111111	"
00 00 10 00	00000000	10101010	00000000	"
00 00 10 01	00000000	10101010	01010101	"
00 00 10 10	00000000	10101010	10101010	"
00 00 10 11	00000000	10101010	11111111	"
00 00 11 00	00000000	11111111	00000000	"
00 00 11 01	00000000	11111111	01010101	"
00 00 11 10	00000000	11111111	10101010	"
00 00 11 11	00000000	11111111	11111111	"
00 01 00 00	01010101	00000000	00000000	"

50

55

5	00 01 00 01	01010101	00000000	01010101	"
	00 01 00 10	01010101	00000000	10101010	"
	00 01 00 11	01010101	00000000	11111111	"
	00 01 01 00	01010101	01010101	00000000	"
	00 01 01 01	01010101	01010101	01010101	"
	00 01 01 10	01010101	01010101	10101010	"
	00 01 01 11	01010101	01010101	11111111	"
10	00 01 10 00	01010101	10101010	00000000	"
	00 01 10 01	01010101	10101010	01010101	"
	00 01 10 10	01010101	10101010	10101010	"
	00 01 10 11	01010101	10101010	11111111	"
	00 01 11 00	01010101	11111111	00000000	"
	00 01 11 01	01010101	11111111	01010101	"
	00 01 11 10	01010101	11111111	10101010	"
	00 01 11 11	01010101	11111111	11111111	"
15	00 10 00 00	10101010	00000000	00000000	"
	00 10 00 01	10101010	00000000	01010101	"
	00 10 00 10	10101010	00000000	10101010	"
	00 10 00 11	10101010	00000000	11111111	"
	00 10 01 00	10101010	01010101	00000000	"
	00 10 01 01	10101010	01010101	01010101	"
	00 10 01 10	10101010	01010101	10101010	"
	00 10 01 11	10101010	01010101	11111111	"
	00 10 10 00	10101010	10101010	00000000	"
	00 10 10 01	10101010	10101010	01010101	"
	00 10 10 10	10101010	10101010	10101010	"
	00 10 10 11	10101010	10101010	11111111	"
	00 10 11 00	10101010	11111111	00000000	"
	00 10 11 01	10101010	11111111	01010101	"
25	00 10 11 10	10101010	11111111	10101010	"
	00 10 11 11	10101010	11111111	11111111	"
	00 11 00 00	11111111	00000000	00000000	"
	00 11 00 01	11111111	00000000	01010101	"
	00 11 00 10	11111111	00000000	10101010	"
	00 11 00 11	11111111	00000000	11111111	"
	00 11 01 00	11111111	01010101	00000000	"
	00 11 01 01	11111111	01010101	01010101	"
30	00 11 01 10	11111111	01010101	10101010	"
	00 11 01 11	11111111	01010101	11111111	"
	00 11 10 00	11111111	10101010	00000000	"
	00 11 10 01	11111111	10101010	01010101	"
	00 11 10 10	11111111	10101010	10101010	"
	00 11 10 11	11111111	10101010	11111111	"
35	00 11 11 00	11111111	11111111	00000000	"
	00 11 11 01	11111111	11111111	01010101	"
	00 11 11 10	11111111	11111111	10101010	"
	00 11 11 11	11111111	11111111	11111111	"
	01 00 00 00	10000000	10000000	10000000 (from indexed surface	"
	01 00 00 01	10000000	10000000	10000000	"
	01 00 00 10	10000000	10000000	10000000	"
	01 00 00 11	10000000	10000000	10000000	"
40	01 00 01 00	10000000	10000000	10000000	"
	01 00 01 01	10000000	10000000	10000000	"
	01 00 01 10	10000000	10000000	10000000	"
	01 00 01 11	10000000	10000000	10000000	"
	01 00 10 00	10000000	10000000	10000000	"

45

50

55

	01 00 10 01	10000000	10000000	10000000	00
	01 00 10 10	10000000	10000000	10000000	00
	01 00 10 11	10000000	10000000	10000000	00
5	01 00 11 00	10000000	10000000	10000000	00
	01 00 11 01	10000000	10000000	10000000	00
	01 00 11 10	10000000	10000000	10000000	00
	01 00 11 11	10000000	10000000	10000000	00
	01 01 00 00	10000000	10000000	10000000	00
	01 01 00 01	10000000	10000000	10000000	00
10	01 01 00 10	10000000	10000000	10000000	00
	01 01 00 11	10000000	10000000	10000000	00
	01 01 01 00	10000000	10000000	10000000	00
	01 01 01 01	10000000	10000000	10000000	00
	01 01 01 10	10000000	10000000	10000000	00
	01 01 01 11	10000000	10000000	10000000	00
15	01 01 10 00	10000000	10000000	10000000	00
	01 01 10 01	10000000	10000000	10000000	00
	01 01 10 10	10000000	10000000	10000000	00
	01 01 10 11	10000000	10000000	10000000	00
	01 01 11 00	10000000	10000000	10000000	00
20	01 01 11 01	10000000	10000000	10000000	00
	01 01 11 10	10000000	10000000	10000000	00
	01 01 11 11	10000000	10000000	10000000	00
	01 10 00 00	10000000	10000000	10000000	00
	01 10 00 01	10000000	10000000	10000000	00
	01 10 00 10	10000000	10000000	10000000	00
25	01 10 00 11	10000000	10000000	10000000	00
	01 10 01 00	10000000	10000000	10000000	00
	01 10 01 01	10000000	10000000	10000000	00
	01 10 01 10	10000000	10000000	10000000	00
	01 10 01 11	10000000	10000000	10000000	00
	01 10 10 00	10000000	10000000	10000000	00
30	01 10 10 01	10000000	10000000	10000000	00
	01 10 10 10	10000000	10000000	10000000	00
	01 10 10 11	10000000	10000000	10000000	00
	01 10 11 00	10000000	10000000	10000000	00
	01 10 11 01	10000000	10000000	10000000	00
	01 10 11 10	10000000	10000000	10000000	00
35	01 10 11 11	10000000	10000000	10000000	00
	01 11 00 00	10000000	10000000	10000000	00
	01 11 00 01	10000000	10000000	10000000	00
	01 11 00 10	10000000	10000000	10000000	00
	01 11 00 11	10000000	10000000	10000000	00
40	01 11 01 00	10000000	10000000	10000000	00
	01 11 01 01	10000000	10000000	10000000	00
	01 11 01 10	10000000	10000000	10000000	00
	01 11 01 11	10000000	10000000	10000000	00
	01 11 10 00	10000000	10000000	10000000	00
	01 11 10 01	10000000	10000000	10000000	00
45	01 11 10 10	10000000	10000000	10000000	00
	01 11 10 11	10000000	10000000	10000000	00
	01 11 11 00	10000000	10000000	10000000	00
	01 11 11 01	10000000	10000000	10000000	00
	01 11 11 10	10000000	10000000	10000000	00
	01 11 11 11	10000000	10000000	10000000	00
50	10 00 00 00	11111111	11111111	00000000	00

	10 00 00 01	11111111	11111111	00000000	"
	10 00 00 10	11111111	11111111	00000000	"
	10 00 00 11	11111111	11111111	00000000	"
5	10 00 01 00	11111111	11111111	00000000	"
	10 00 01 01	11111111	11111111	00000000	"
	10 00 01 10	11111111	11111111	00000000	"
	10 00 01 11	11111111	11111111	00000000	"
	10 00 10 00	11111111	11111111	00000000	"
	10 00 10 01	11111111	11111111	00000000	"
10	10 00 10 10	11111111	11111111	00000000	"
	10 00 10 11	11111111	11111111	00000000	"
	10 00 11 00	11111111	11111111	00000000	"
	10 00 11 01	11111111	11111111	00000000	"
	10 00 11 10	11111111	11111111	00000000	"
	10 00 11 11	11111111	11111111	00000000	"
15	10 01 00 00	11111111	11111111	00000000	"
	10 01 00 01	11111111	11111111	00000000	"
	10 01 00 10	11111111	11111111	00000000	"
	10 01 00 11	11111111	11111111	00000000	"
	10 01 01 00	11111111	11111111	00000000	"
20	10 01 01 01	11111111	11111111	00000000	"
	10 01 01 10	11111111	11111111	00000000	"
	10 01 01 11	11111111	11111111	00000000	"
	10 01 10 00	11111111	11111111	00000000	"
	10 01 10 01	11111111	11111111	00000000	"
	10 01 10 10	11111111	11111111	00000000	"
25	10 01 10 11	11111111	11111111	00000000	"
	10 01 11 00	11111111	11111111	00000000	"
	10 01 11 01	11111111	11111111	00000000	"
	10 01 11 10	11111111	11111111	00000000	"
	10 01 11 11	11111111	11111111	00000000	"
30	10 10 00 00	11111111	11111111	00000000	"
	10 10 00 01	11111111	11111111	00000000	"
	10 10 00 10	11111111	11111111	00000000	"
	10 10 00 11	11111111	11111111	00000000	"
	10 10 01 00	11111111	11111111	00000000	"
	10 10 01 01	11111111	11111111	00000000	"
35	10 10 01 10	11111111	11111111	00000000	"
	10 10 01 11	11111111	11111111	00000000	"
	10 10 10 00	11111111	11111111	00000000	"
	10 10 10 01	11111111	11111111	00000000	"
	10 10 10 10	11111111	11111111	00000000	"
	10 10 10 11	11111111	11111111	00000000	"
40	10 10 11 00	11111111	11111111	00000000	"
	10 10 11 01	11111111	11111111	00000000	"
	10 10 11 10	11111111	11111111	00000000	"
	10 10 11 11	11111111	11111111	00000000	"
	10 11 00 00	11111111	11111111	00000000	"
45	10 11 00 01	11111111	11111111	00000000	"
	10 11 00 10	11111111	11111111	00000000	"
	10 11 00 11	11111111	11111111	00000000	"
	10 11 01 00	11111111	11111111	00000000	"
	10 11 01 01	11111111	11111111	00000000	"
	10 11 01 10	11111111	11111111	00000000	"
50	10 11 01 11	11111111	11111111	00000000	"
	10 11 10 00	11111111	11111111	00000000	"

	10	11	10	01	11111111	11111111	00000000	"
	10	11	10	10	11111111	11111111	00000000	"
	10	11	10	11	11111111	11111111	00000000	"
5	10	11	11	00	11111111	11111111	00000000	"
	10	11	11	01	11111111	11111111	00000000	"
	10	11	11	10	11111111	11111111	00000000	"
	10	11	11	11	11111111	11111111	00000000	"
	11	00	00	00	00000000	10000000	11111111	"
10	11	00	00	01	00000000	10000000	11111111	"
	11	00	00	10	00000000	10000000	11111111	"
	11	00	00	11	00000000	10000000	11111111	"
	11	00	01	00	00000000	10000000	11111111	"
	11	00	01	01	00000000	10000000	11111111	"
	11	00	01	10	00000000	10000000	11111111	"
15	11	00	01	11	00000000	10000000	11111111	"
	11	00	10	00	00000000	10000000	11111111	"
	11	00	10	01	00000000	10000000	11111111	"
	11	00	10	10	00000000	10000000	11111111	"
	11	00	10	11	00000000	10000000	11111111	"
	11	00	11	00	00000000	10000000	11111111	"
20	11	00	11	01	00000000	10000000	11111111	"
	11	00	11	10	00000000	10000000	11111111	"
	11	00	11	11	00000000	10000000	11111111	"
	11	01	00	00	00000000	10000000	11111111	"
	11	01	00	01	00000000	10000000	11111111	"
	11	01	00	10	00000000	10000000	11111111	"
25	11	01	00	11	00000000	10000000	11111111	"
	11	01	01	00	00000000	10000000	11111111	"
	11	01	01	01	00000000	10000000	11111111	"
	11	01	01	10	00000000	10000000	11111111	"
	11	01	01	11	00000000	10000000	11111111	"
30	11	01	10	00	00000000	10000000	11111111	"
	11	01	10	01	00000000	10000000	11111111	"
	11	01	10	10	00000000	10000000	11111111	"
	11	01	10	11	00000000	10000000	11111111	"
	11	01	11	00	00000000	10000000	11111111	"
	11	01	11	01	00000000	10000000	11111111	"
35	11	01	11	10	00000000	10000000	11111111	"
	11	01	11	11	00000000	10000000	11111111	"
	11	10	00	00	00000000	10000000	11111111	"
	11	10	00	01	00000000	10000000	11111111	"
	11	10	00	10	00000000	10000000	11111111	"
	11	10	00	11	00000000	10000000	11111111	"
40	11	10	01	00	00000000	10000000	11111111	"
	11	10	01	01	00000000	10000000	11111111	"
	11	10	01	10	00000000	10000000	11111111	"
	11	10	01	11	00000000	10000000	11111111	"
	11	10	10	00	00000000	10000000	11111111	"
45	11	10	10	01	00000000	10000000	11111111	"
	11	10	10	10	00000000	10000000	11111111	"
	11	10	10	11	00000000	10000000	11111111	"
	11	10	11	00	00000000	10000000	11111111	"
	11	10	11	01	00000000	10000000	11111111	"
	11	10	11	10	00000000	10000000	11111111	"
50	11	10	11	11	00000000	10000000	11111111	"
	11	11	00	00	00000000	10000000	11111111	"

```

5      11 11 00 01      00000000      10000000      11111111      "
      11 11 00 10      00000000      10000000      11111111      "
      11 11 00 11      00000000      10000000      11111111      "
      11 11 01 00      00000000      10000000      11111111      "
      11 11 01 01      00000000      10000000      11111111      "
      11 11 01 10      00000000      10000000      11111111      "
      11 11 01 11      00000000      10000000      11111111      "
10     11 11 10 00      00000000      10000000      11111111      "
      11 11 10 01      00000000      10000000      11111111      "
      11 11 10 10      00000000      10000000      11111111      "
      11 11 10 11      00000000      10000000      11111111      "
      11 11 11 00      00000000      10000000      11111111      "
      11 11 11 01      00000000      10000000      11111111      "
      11 11 11 10      00000000      10000000      11111111      "
15     11 11 11 11      00000000      10000000      11111111      "

```

Case 2. Assume that we want the true-color surface to take priority over the indexed color surface. Then the hardware color map would be populated as:

bit numbers
76 54 32 10

```

-----
25     color map      color map data
      address      R      G      B      (remarks)
-----
      00 00 00 00      00000000      00000000      00000000      (background color)
      00 00 00 01      00000000      00000000      01010101      (from true color surf
      00 00 00 10      00000000      00000000      10101010      "
      00 00 00 11      00000000      00000000      11111111      "
      00 00 01 00      00000000      01010101      11111111      "
30     00 00 01 01      00000000      01010101      11111111      "
      00 00 01 10      00000000      01010101      11111111      "
      00 00 01 11      00000000      01010101      11111111      "
      00 00 10 00      00000000      10101010      00000000      "
      00 00 10 01      00000000      10101010      01010101      "
      00 00 10 10      00000000      10101010      10101010      "
      00 00 10 11      00000000      10101010      11111111      "
35     00 00 11 00      00000000      11111111      00000000      "
      00 00 11 01      00000000      11111111      01010101      "
      00 00 11 10      00000000      11111111      10101010      "
      00 00 11 11      00000000      11111111      11111111      "
      00 01 00 00      01010101      00000000      00000000      "
      00 01 00 01      01010101      00000000      01010101      "
      00 01 00 10      01010101      00000000      10101010      "
40     00 01 00 11      01010101      00000000      11111111      "
      00 01 01 00      01010101      01010101      00000000      "
      00 01 01 01      01010101      01010101      01010101      "
      00 01 01 10      01010101      01010101      10101010      "
      00 01 01 11      01010101      01010101      11111111      "
      00 01 10 00      01010101      10101010      00000000      "
      00 01 10 01      01010101      10101010      01010101      "

```

5	00 01 10 10	01010101	10101010	10101010	"
	00 01 10 11	01010101	10101010	11111111	"
	00 01 11 00	01010101	11111111	00000000	"
	00 01 11 01	01010101	11111111	01010101	"
	00 01 11 10	01010101	11111111	10101010	"
	00 01 11 11	01010101	11111111	11111111	"
10	00 10 00 00	10101010	00000000	00000000	"
	00 10 00 01	10101010	00000000	01010101	"
	00 10 00 10	10101010	00000000	10101010	"
	00 10 00 11	10101010	00000000	11111111	"
	00 10 01 00	10101010	01010101	00000000	"
	00 10 01 01	10101010	01010101	01010101	"
	00 10 01 10	10101010	01010101	10101010	"
15	00 10 01 11	10101010	01010101	11111111	"
	00 10 10 00	10101010	10101010	00000000	"
	00 10 10 01	10101010	10101010	01010101	"
	00 10 10 10	10101010	10101010	10101010	"
	00 10 10 11	10101010	10101010	11111111	"
	00 10 11 00	10101010	11111111	00000000	"
	00 10 11 01	10101010	11111111	01010101	"
20	00 10 11 10	10101010	11111111	10101010	"
	00 10 11 11	10101010	11111111	11111111	"
	00 11 00 00	11111111	00000000	00000000	"
	00 11 00 01	11111111	00000000	01010101	"
	00 11 00 10	11111111	00000000	10101010	"
	00 11 00 11	11111111	00000000	11111111	"
	00 11 01 00	11111111	01010101	00000000	"
25	00 11 01 01	11111111	01010101	01010101	"
	00 11 01 10	11111111	01010101	10101010	"
	00 11 01 11	11111111	01010101	11111111	"
	00 11 10 00	11111111	10101010	00000000	"
	00 11 10 01	11111111	10101010	01010101	"
	00 11 10 10	11111111	10101010	10101010	"
	00 11 10 11	11111111	10101010	11111111	"
	00 11 11 00	11111111	11111111	00000000	"
30	00 11 11 01	11111111	11111111	01010101	"
	00 11 11 10	11111111	11111111	10101010	"
	00 11 11 11	11111111	11111111	11111111	"
	01 00 00 00	10000000	10000000	10000000 (from indexed surface	
	01 00 00 01	00000000	00000000	01010101 (from true color sur-	
	01 00 00 10	00000000	00000000	10101010	"
35	01 00 00 11	00000000	00000000	11111111	"
	01 00 01 00	00000000	01010101	11111111	"
	01 00 01 01	00000000	01010101	11111111	"
	01 00 01 10	00000000	01010101	11111111	"
	01 00 01 11	00000000	01010101	11111111	"
	01 00 10 00	00000000	10101010	00000000	"
	01 00 10 01	00000000	10101010	01010101	"
	01 00 10 10	00000000	10101010	10101010	"
40	01 00 10 11	00000000	10101010	11111111	"
	01 00 11 00	00000000	11111111	00000000	"
	01 00 11 01	00000000	11111111	01010101	"
	01 00 11 10	00000000	11111111	10101010	"
	01 00 11 11	00000000	11111111	11111111	"
	01 01 00 00	01010101	00000000	00000000	"
	01 01 00 01	01010101	00000000	01010101	"

45

50

55

Line	Address	Hex	ASCII
5	01 01 00 10	01010101	
	01 01 00 11	01010101	
	01 01 01 00	01010101	
	01 01 01 01	01010101	
	01 01 01 10	01010101	
	01 01 01 11	01010101	
10	01 01 10 00	01010101	
	01 01 10 01	01010101	
	01 01 10 10	01010101	
	01 01 10 11	01010101	
	01 01 11 00	01010101	
	01 01 11 01	01010101	
	01 01 11 10	01010101	
	01 01 11 11	01010101	
15	01 10 00 00	10101010	
	01 10 00 01	10101010	
	01 10 00 10	10101010	
	01 10 00 11	10101010	
	01 10 01 00	10101010	
	01 10 01 01	10101010	
	01 10 01 10	10101010	
	01 10 01 11	10101010	
20	01 10 10 00	10101010	
	01 10 10 01	10101010	
	01 10 10 10	10101010	
	01 10 10 11	10101010	
	01 10 11 00	10101010	
	01 10 11 01	10101010	
	01 10 11 10	10101010	
	01 10 11 11	10101010	
25	01 11 00 00	11111111	
	01 11 00 01	11111111	
	01 11 00 10	11111111	
	01 11 00 11	11111111	
	01 11 01 00	11111111	
	01 11 01 01	11111111	
	01 11 01 10	11111111	
	01 11 01 11	11111111	
30	01 11 10 00	11111111	
	01 11 10 01	11111111	
	01 11 10 10	11111111	
	01 11 10 11	11111111	
	01 11 11 00	11111111	
	01 11 11 01	11111111	
	01 11 11 10	11111111	
	01 11 11 11	11111111	
	10 00 00 00	11111111	
	10 00 00 01	00000000	(from indexed surface
	10 00 00 10	00000000	(from true color surf
	10 00 00 11	00000000	
40	10 00 01 00	00000000	
	10 00 01 01	00000000	
	10 00 01 10	00000000	
	10 00 01 11	00000000	
	10 00 10 00	00000000	
	10 00 10 01	00000000	

	10 00 10 10	00000000	10101010	10101010	"
	10 00 10 11	00000000	10101010	11111111	"
5	10 00 11 00	00000000	11111111	00000000	"
	10 00 11 01	00000000	11111111	01010101	"
	10 00 11 10	00000000	11111111	10101010	"
	10 00 11 11	00000000	11111111	11111111	"
	10 01 00 00	01010101	00000000	00000000	"
	10 01 00 01	01010101	00000000	01010101	"
10	10 01 00 10	01010101	00000000	10101010	"
	10 01 00 11	01010101	00000000	11111111	"
	10 01 01 00	01010101	01010101	00000000	"
	10 01 01 01	01010101	01010101	01010101	"
	10 01 01 10	01010101	01010101	10101010	"
	10 01 01 11	01010101	01010101	11111111	"
	10 01 10 00	01010101	10101010	00000000	"
15	10 01 10 01	01010101	10101010	01010101	"
	10 01 10 10	01010101	10101010	10101010	"
	10 01 10 11	01010101	10101010	11111111	"
	10 01 11 00	01010101	11111111	00000000	"
	10 01 11 01	01010101	11111111	01010101	"
	10 01 11 10	01010101	11111111	10101010	"
	10 01 11 11	01010101	11111111	11111111	"
20	10 10 00 00	10101010	00000000	00000000	"
	10 10 00 01	10101010	00000000	01010101	"
	10 10 00 10	10101010	00000000	10101010	"
	10 10 00 11	10101010	00000000	11111111	"
	10 10 01 00	10101010	01010101	00000000	"
	10 10 01 01	10101010	01010101	01010101	"
25	10 10 01 10	10101010	01010101	10101010	"
	10 10 01 11	10101010	01010101	11111111	"
	10 10 10 00	10101010	10101010	00000000	"
	10 10 10 01	10101010	10101010	01010101	"
	10 10 10 10	10101010	10101010	10101010	"
	10 10 10 11	10101010	10101010	11111111	"
	10 10 11 00	10101010	11111111	00000000	"
30	10 10 11 01	10101010	11111111	01010101	"
	10 10 11 10	10101010	11111111	10101010	"
	10 10 11 11	10101010	11111111	11111111	"
	10 11 00 00	11111111	00000000	00000000	"
	10 11 00 01	11111111	00000000	01010101	"
	10 11 00 10	11111111	00000000	10101010	"
35	10 11 00 11	11111111	00000000	11111111	"
	10 11 01 00	11111111	01010101	00000000	"
	10 11 01 01	11111111	01010101	01010101	"
	10 11 01 10	11111111	01010101	10101010	"
	10 11 01 11	11111111	01010101	11111111	"
	10 11 10 00	11111111	10101010	00000000	"
	10 11 10 01	11111111	10101010	01010101	"
40	10 11 10 10	11111111	10101010	10101010	"
	10 11 10 11	11111111	10101010	11111111	"
	10 11 11 00	11111111	11111111	00000000	"
	10 11 11 01	11111111	11111111	01010101	"
	10 11 11 10	11111111	11111111	10101010	"
	10 11 11 11	11111111	11111111	11111111	"
45	11 00 00 00	00000000	00000000	11111111 (from indexed surfa	
	11 00 00 01	00000000	00000000	01010101 (from true color su	

50

55

	11 00 00 10	00000000	00000000	10101010	"
	11 00 00 11	00000000	00000000	11111111	"
	11 00 01 00	00000000	01010101	11111111	"
5	11 00 01 01	00000000	01010101	11111111	"
	11 00 01 10	00000000	01010101	11111111	"
	11 00 01 11	00000000	01010101	11111111	"
	11 00 10 00	00000000	10101010	00000000	"
	11 00 10 01	00000000	10101010	01010101	"
	11 00 10 10	00000000	10101010	10101010	"
10	11 00 10 11	00000000	10101010	11111111	"
	11 00 11 00	00000000	11111111	00000000	"
	11 00 11 01	00000000	11111111	01010101	"
	11 00 11 10	00000000	11111111	10101010	"
	11 00 11 11	00000000	11111111	11111111	"
15	11 01 00 00	01010101	00000000	00000000	"
	11 01 00 01	01010101	00000000	01010101	"
	11 01 00 10	01010101	00000000	10101010	"
	11 01 00 11	01010101	00000000	11111111	"
	11 01 01 00	01010101	01010101	00000000	"
	11 01 01 01	01010101	01010101	01010101	"
20	11 01 01 10	01010101	01010101	10101010	"
	11 01 01 11	01010101	01010101	11111111	"
	11 01 10 00	01010101	10101010	00000000	"
	11 01 10 01	01010101	10101010	01010101	"
	11 01 10 10	01010101	10101010	10101010	"
	11 01 10 11	01010101	10101010	11111111	"
25	11 01 11 00	01010101	11111111	00000000	"
	11 01 11 01	01010101	11111111	01010101	"
	11 01 11 10	01010101	11111111	10101010	"
	11 01 11 11	01010101	11111111	11111111	"
	11 10 00 00	10101010	00000000	00000000	"
30	11 10 00 01	10101010	00000000	01010101	"
	11 10 00 10	10101010	00000000	10101010	"
	11 10 00 11	10101010	00000000	11111111	"
	11 10 01 00	10101010	01010101	00000000	"
	11 10 01 01	10101010	01010101	01010101	"
	11 10 01 10	10101010	01010101	10101010	"
35	11 10 01 11	10101010	01010101	11111111	"
	11 10 10 00	10101010	10101010	00000000	"
	11 10 10 01	10101010	10101010	01010101	"
	11 10 10 10	10101010	10101010	10101010	"
	11 10 10 11	10101010	10101010	11111111	"
	11 10 11 00	10101010	11111111	00000000	"
40	11 10 11 01	10101010	11111111	01010101	"
	11 10 11 10	10101010	11111111	10101010	"
	11 10 11 11	10101010	11111111	11111111	"
	11 11 00 00	11111111	00000000	00000000	"
	11 11 00 01	11111111	00000000	01010101	"
	11 11 00 10	11111111	00000000	10101010	"
45	11 11 00 11	11111111	00000000	11111111	"
	11 11 01 00	11111111	01010101	00000000	"
	11 11 01 01	11111111	01010101	01010101	"
	11 11 01 10	11111111	01010101	10101010	"
	11 11 01 11	11111111	01010101	11111111	"
50	11 11 10 00	11111111	10101010	00000000	"
	11 11 10 01	11111111	10101010	01010101	"

11 11 10 10 11111111 10101010 10101010 "
 11 11 10 11 11111111 10101010 11111111 "
 11 11 11 00 11111111 11111111 00000000 "
 11 11 11 01 11111111 11111111 01010101 "
 5 11 11 11 10 11111111 11111111 10101010 "
 11 11 11 11 11111111 11111111 11111111 "

10

Claims

1. An apparatus for representing three-component color data in a single index reference system comprising:
 15 processing means responsive to applied data for deriving a single index and a plurality of primary color component values, the applied data representing selected numbers of color levels of color components and representing color component values of a desired color of an image; and
 memory means for storing the derived primary color component values in a sequence such that the single
 20 index references the color component values of the desired color.

25

30

35

40

45

50

55

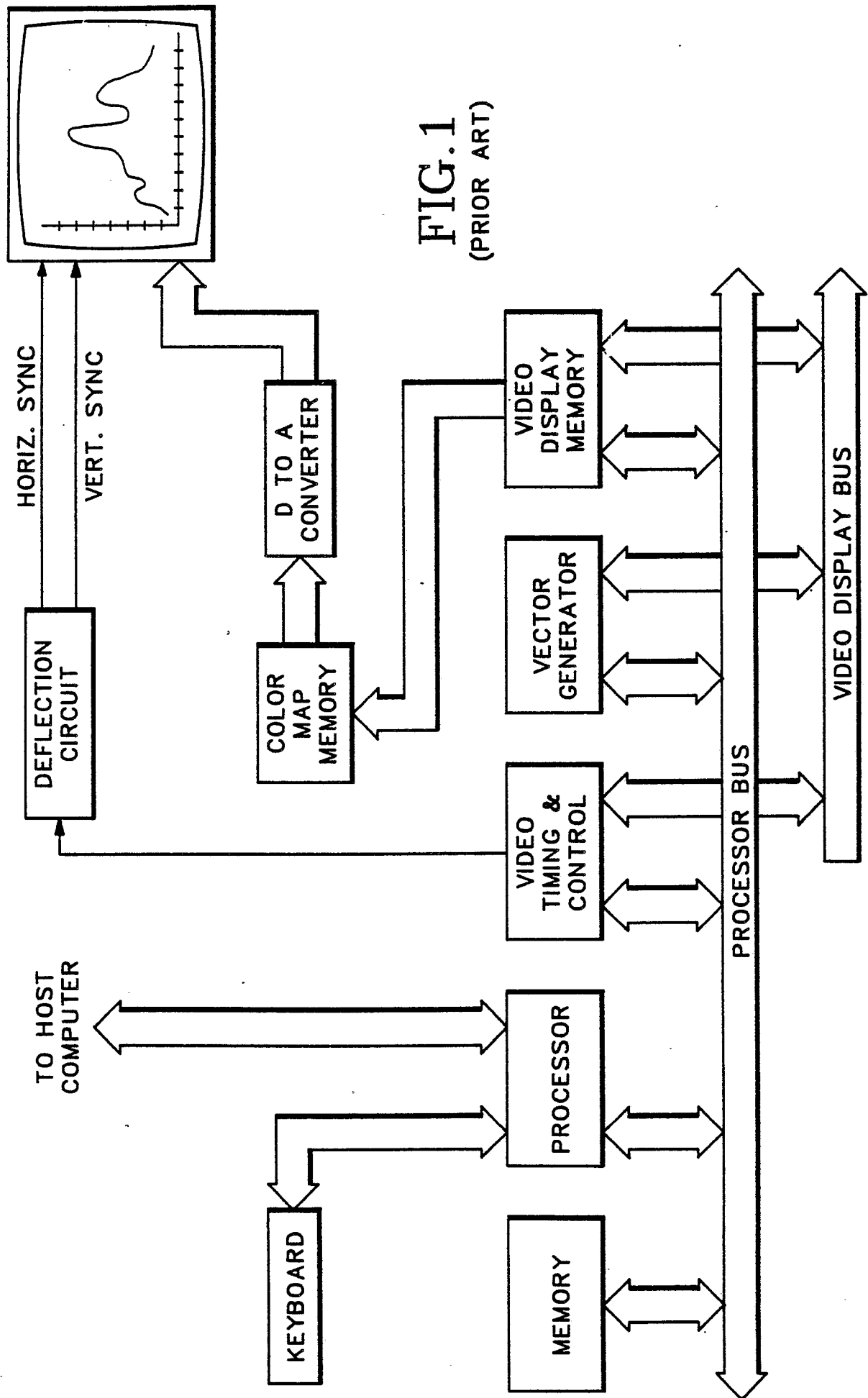
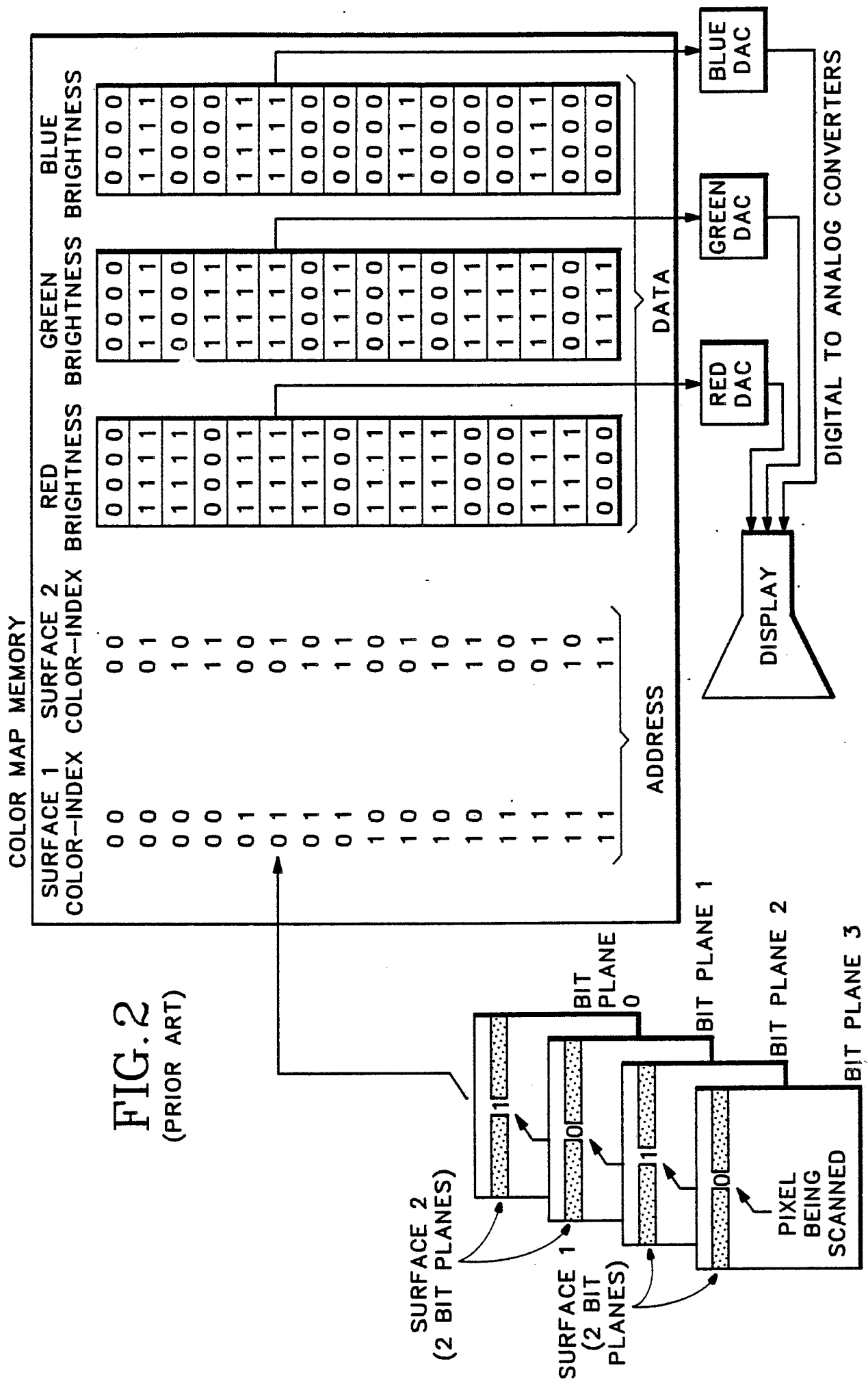


FIG. 1
(PRIOR ART)



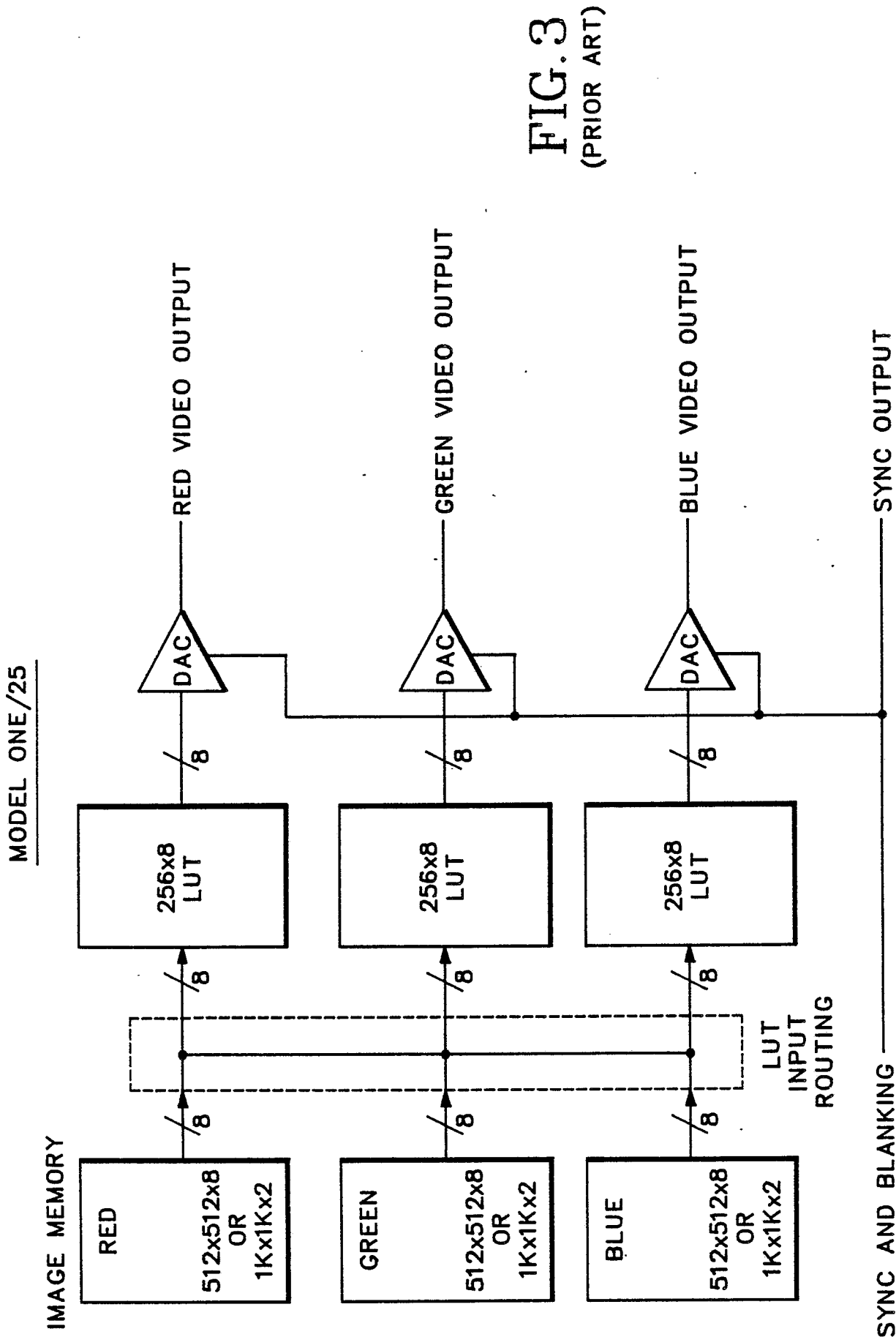


FIG.3
(PRIOR ART)

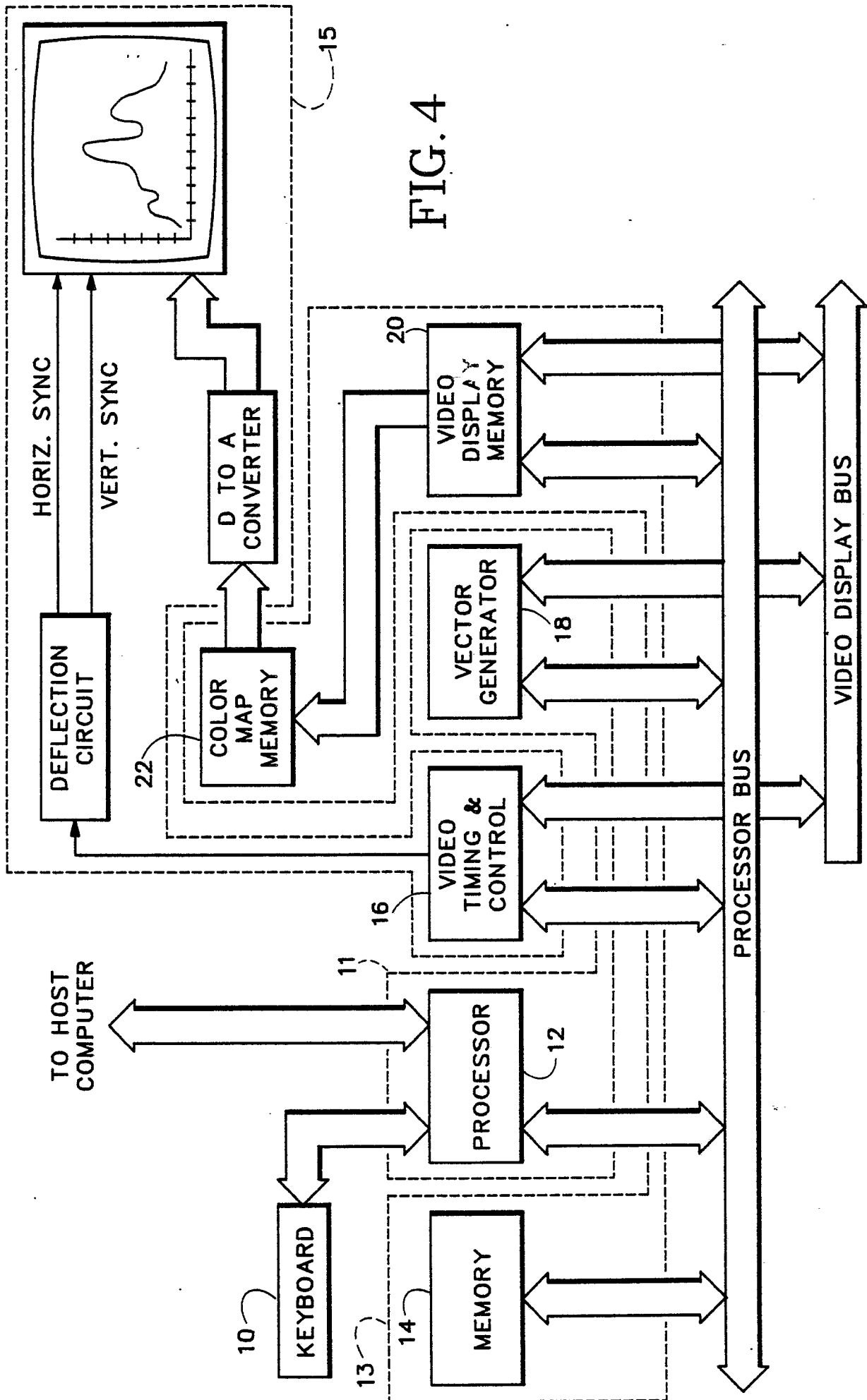


FIG. 4

COLOR MAP MEMORY

RED BRIGHTNESS (DERIVED)	GREEN BRIGHTNESS (DERIVED)	BLUE BRIGHTNESS (DERIVED)
0 1 0 1 0 1 0 1	0 0 0 0 0 0 0 0	0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1	0 0 0 0 0 0 0 0	1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1	0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1
0 1 0 1 0 1 0 1	0 1 0 1 0 1 0 1	0 0 0 0 0 0 0 0
0 1 0 1 0 1 0 1	0 1 0 1 0 1 0 1	0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1	0 1 0 1 0 1 0 1	1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1	0 1 0 1 0 1 0 1	1 1 1 1 1 1 1 1
0 1 0 1 0 1 0 1	1 0 1 0 1 0 1 0	0 0 0 0 0 0 0 0
0 1 0 1 0 1 0 1	1 0 1 0 1 0 1 0	0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1	1 0 1 0 1 0 1 0	1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1	1 0 1 0 1 0 1 0	1 1 1 1 1 1 1 1
0 1 0 1 0 1 0 1	1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0
0 1 0 1 0 1 0 1	1 1 1 1 1 1 1 1	0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1	1 1 1 1 1 1 1 1	1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1
1 0 1 0 1 0 1 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
1 0 1 0 1 0 1 0	0 0 0 0 0 0 0 0	0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0	0 0 0 0 0 0 0 0	1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0	0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1
1 0 1 0 1 0 1 0	0 1 0 1 0 1 0 1	0 0 0 0 0 0 0 0
1 0 1 0 1 0 1 0	0 1 0 1 0 1 0 1	0 1 0 1 0 1 0 1

00 01 10 11

↑

FIG. 5

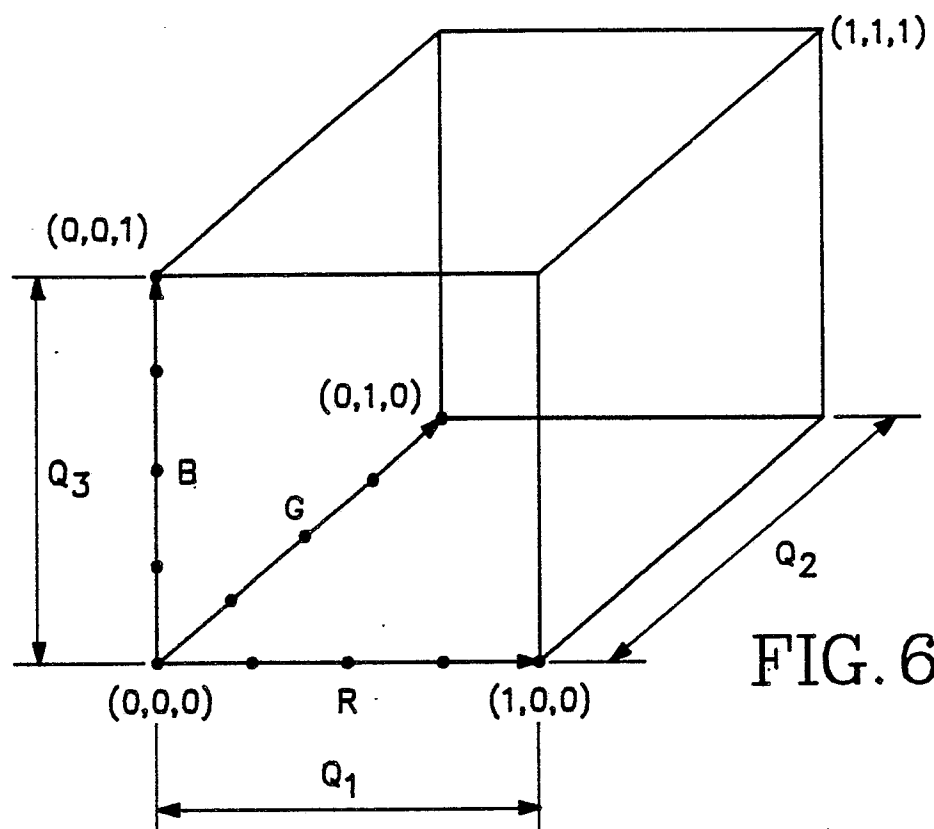


FIG. 6

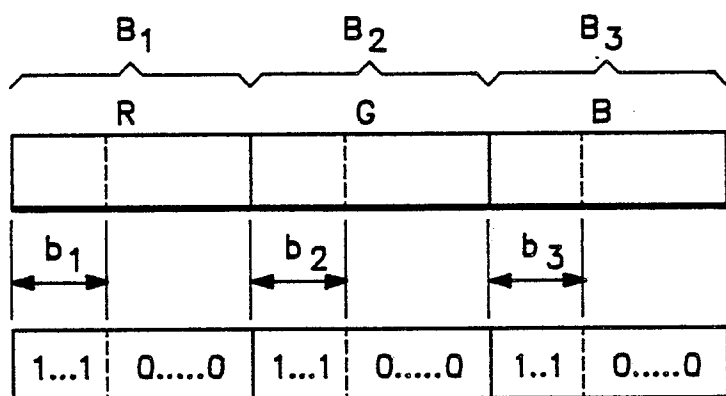


FIG. 7

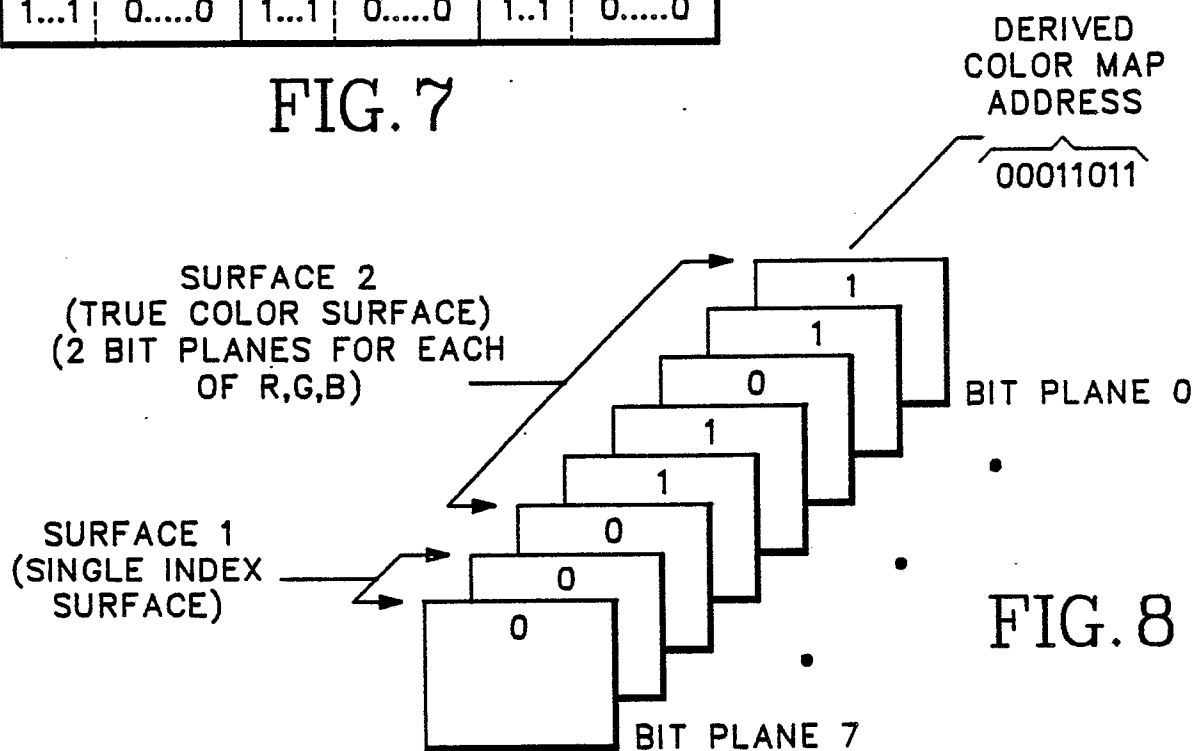


FIG. 8



DOCUMENTS CONSIDERED TO BE RELEVANT			EP 88115124.5
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int. Cl.4)
D, A	<u>US - A - 4 509 043</u> (MOSSAIDES) * Abstract; fig. 8 * --	1	G 09 G 1/28 H 04 N 5/262 H 04 N 9/74
A	<u>US - A - 4 578 673</u> (YIANILOS) --		
A	<u>EP - A2 - 0 165 441</u> (IBM) --		
A	<u>EP - A2 - 0 105 724</u> (FANUC) ----		
The present search report has been drawn up for all claims			TECHNICAL FIELDS SEARCHED (Int. Cl.4) G 09 G 1/00 H 04 N 5/00 H 04 N 9/00 H 04 N 15/00
Place of search VIENNA		Date of completion of the search 18-01-1989	Examiner BENISCHKA
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			