Michael A. Covington
Color vision and the VGA.
PC Techniques 1.2:34-39 (June/July 1990).

Please note that this material *does not* apply directly to present-day computers, and the software *may not work* under current versions of Windows. It has not been tested under Windows 95 or any later version. *The author is not prepared to do any further work on this* unless someone employs him as a consultant.

For an up-to-date survey of color vision as applied to computer graphics and other modern technologies, see *Introduction to Color Imaging Science,* by Hsien-Che Lee (Cambridge University Press, 2005).

Building Skills in PC Software Development

$4.95 ($5.95 CANADA)

PC TECHNIQUES

JUNE/JULY 1990
VOL. 1, NO. 2

VIVA VIDEO!

VGA Analog Color

Meet The 8514/A

C Drawing Tools

BASICs Reborn

More HAX

Color Vision and the VGA

Michael Covington

Confused by the maze of VGA colors? This article shows you how to use the VGA's 262,144 colors.

Three years ago, color on the PC suddenly got a *great* deal better, with IBM's introduction of the VGA. By 1990, the VGA has come down in price to the point where many (if not most) PC owners have it installed. Rather than being just a curiosity, VGA analog color is now a feature that can be exploited in your applications to produce effects simply not possible with earlier EGA-style 16-color video. The VGA offers you 262,144 colors—an explosion of choices that requires your understanding a little bit about human vision and color theory. Once you understand the nature of color and how it affects the human eye, dealing with VGA analog color isn't a great deal more than stuffing values in registers. Let's take a closer (and full-color) look before we create the color-mixing program shown in Color Plate 1.

What is color?—Generally speaking, color is the wavelength of light. Red light has a longer wavelength than green, which has a longer wavelength than blue. Color in this sense is technically called *hue*. Colors also have *intensity* and *saturation*. Intensity is the amount of light reaching your eye. There's a minimum intensity—black, corresponding to no light at all—but there's no maximum intensity. Saturation, in turn, is what distinguishes strong from weak colors—red from pink, for instance. To reduce the saturation of a color, you mix it with white or gray. The minimum saturation is white or gray; the maximum saturation is monochromatic (single-wavelength) light, such as the light from a laser or a nearly monochromatic LED.

Color Plate 1

To visualize hue, intensity, and saturation, imagine there's a red LED in front of you. If you make it brighter or dimmer, you're changing the intensity. No matter how bright you make it, it won't be pink—which is why it's misleading to describe pink as "light red." To go from red to pink you'd have to mix in some white light, which is a change in saturation. And if you transformed the red LED into a green one, you'd be changing the hue.

We're all color blind—I said color was wavelength. That's not quite true. Some colors—the purples and magentas—do not occur in the spectrum; no single wavelength is ever seen as magenta. The eye sees purples and magentas only when it receives a mixture of wavelengths.

Color vision is a highly ambiguous process. If I show you light from the yellow part of the spectrum, you will see yellow. If I show you an equal mixture of red and green light, you will also see it as yellow, even though there's no yellow in it. That is, two light sources with completely different wavelengths—one pure, the other a mixture—will look alike to you.

In fact, any hue that the eye can see can be imitated by mixing the

Color Plate 2

three *primary colors*, red, green, and blue. (Color Plate 2 shows how the PC uses this system to generate color.) Color photography and color video are possible because of precisely this.

Severely color-blind people need only two primaries. They can match every hue that they see by mixing just *two* suitably chosen colors, usually yellow and blue; red and green aren't necessary. In milder cases, the color-blind person still needs three primaries, but one of them is less important than it ought to be—small differences in the amount of it aren't noticed.

Mild color blindness is very common. While preparing this article I discovered that one of my eyes is more sensitive to red-green distinctions than the other, although both are in the normal range. Now I don't know which eye sees the way other people do!

There could well be a race of aliens whose eyes analyze the spectrum into four, or ten, or a hundred primaries. They would consider all of us color blind.

Mixing colors—The familiar artist's color wheel (Figure 1) displays the range of perceptible hues in a circle. It consists of the spectrum from blue to red, wrapped around until its ends almost meet, with the ends connected by purples and magentas.

Artists use color wheels to figure out how to mix paint. I said earlier that the primary colors were red, green, and blue. That's right as long as you're mixing light sources, but the primaries for mixing paint are different. The reason is that when you mix two lights, the result contains all the wavelengths emitted by *either* source, but when you mix paint, the result contains only the wavelengths reflected by *both* pigments. That's why the primaries for paint are yellow, magenta, and cyan—often approximated by yellow, red, and blue.

If you mix yellow and magenta paint, you get a range of oranges and reds—exactly what's between yellow and magenta on the color wheel. Likewise, if you mix blue and green light, you get cyan—which is between blue and green on the wheel. So far, so good.

But color wheels are based on artists' subjective impressions, and for greater accuracy, they should be replaced with something more scientific. That's where the CIE chromaticity chart (Figure 2) comes in. It's based on measurements of the levels of red, green, and blue light that people need to match observed hues.

The corners of the chart are mathematical abstractions—extreme colors that can never exist in reality. The spectrum is horseshoe-shaped, and the non-spectral magentas are on a line linking its ends. This horseshoe encloses all perceptible colors. Just as in the color wheel, saturation goes down as you approach the middle; the exact center is gray or white. Figure 3 shows where some familiar light sources fall on the chart.

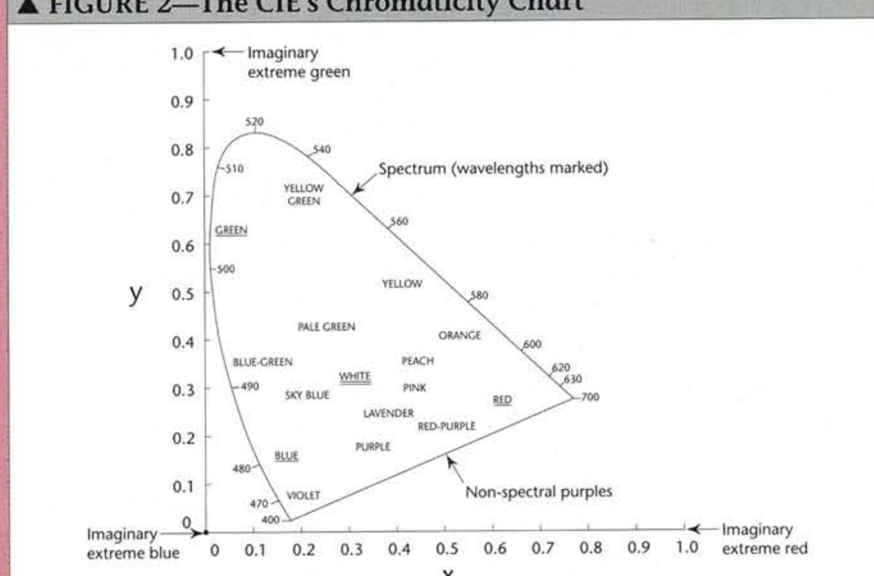
Why didn't we print the chart in color? Simple. The inks available to us aren't saturated enough. You will never see anything on a printed page—or on your computer screen—that looks exactly like a red LED. There just isn't a red ink or video phosphor that's saturated enough.

And here's a crucial point. Mixing three colors can only produce the colors that are *between* the corresponding three points on the chromaticity chart. If the original three sources are widely separated, you can get all hues, but not all saturations. That's why some bright colors will never be seen on video screens. It's also why artists constantly beg for better pigments. Dull yellow plus dull cyan will never yield bright green.

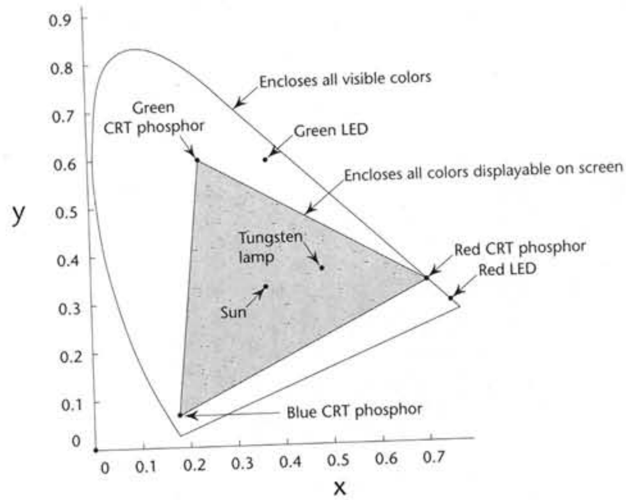
How the PC makes colors—The face of a color CRT is covered with tiny

▲ FIGURE 1—The Artist's Color Wheel

▲ FIGURE 2—The CIE's Chromaticity Chart



▲ FIGURE 3—Light Sources Plotted on the CIE Chart



▲ TABLE 1—CGA Color Mixing

Color Number		I R G B	Color
0	=	0 0 0 0	black
1	=	0 0 0 1	blue
2	=	0 0 1 0	green
3	=	0 0 1 1	cyan (blue + green)
4	=	0 1 0 0	red
5	=	0 1 0 1	magenta (red + blue)
6	=	0 1 1 0	brown (red + green)
7	=	0 1 1 1	gray
8	=	1 0 0 0	dark gray (intensified black)
9	=	1 0 0 1	intensified blue
10	=	1 0 1 0	intensified green
11	=	1 0 1 1	intensified cyan
12	=	1 1 0 0	intensified red
13	=	1 1 0 1	intensified magenta
14	=	1 1 1 0	intensified brown (= yellow)
15	=	1 1 1 1	intensified gray (= white)

▲ TABLE 2—EGA Color Mixing

Color Number	Color Register	r g b R G B	Color
0	0 =	0 0 0 0 0 0	black
1	1 =	0 0 0 0 0 1	blue
2	2 =	0 0 0 0 1 0	green
3	3 =	0 0 0 0 1 1	cyan (blue + green)
4	4 =	0 0 0 1 0 0	red
5	5 =	0 0 0 1 0 1	magenta (red + blue)
6	20 =	0 1 0 1 0 0	brown (red + green)
7	7 =	0 0 0 1 1 1	gray
8	56 =	1 1 1 0 0 0	dark gray (intensified black)
9	57 =	1 1 1 0 0 1	intensified blue
10	58 =	1 1 1 0 1 0	intensified green
11	59 =	1 1 1 0 1 1	intensified cyan
12	60 =	1 1 1 1 0 0	intensified red
13	61 =	1 1 1 1 0 1	intensified magenta
14	62 =	1 1 1 1 1 0	intensified brown (= yellow)
15	63 =	1 1 1 1 1 1	intensified gray

red, green, and blue dots (Color Plate 2), or occasionally stripes, that can be illuminated by varying the strength of the electron beam. The dots are so close together that the eye cannot distinguish them. Instead, the light from adjacent dots mixes and creates a combined color sensation. For example, if the red and green dots are illuminated, you will see yellow. (The dots are not pixels; ideally, they should be smaller than pixels, so that each pixel covers several of them.)

The way the CGA controls the dots is simple (Table 1). The CGA's colors are numbered 0 to 15—that's binary 0000 to 1111—and the four bits of the color number, in binary, control which dots are illuminated. The three low-order bits control red, green, and blue respectively, and the high bit, labeled *intensify*, tells the hardware to make all three dots brighter. This is sometimes called the IRGB system. For example, yellow is intensified red plus green, I=1, R=1, G=1, B=0, which is color code IRGB 1110, or 14.

Instead of four bits per color, the EGA uses six, in two sets of three: weak red, green, and blue, and strong red, green, and blue. This is abbreviated *rgbRGB*. There are four possible brightnesses for each dot: off, weak, strong, or weak plus strong. For example, brown is weak green plus strong red. This gives a total of 64 possible colors.

Register madness—The EGA also introduced *color registers* to map the color numbers onto the actual colors. After all, there are still only four bits per pixel, and hence only 16 colors usable at a time. For each of the 16 colors, the EGA has a register in which the actual color definition is stored.

Table 2 shows how this works. Color 6, for example, is normally brown because color register 6 defines it as weak green plus strong red, a combination that looks brown on the screen. The definition is expressed in rgbRGB format as the binary number 010100, equivalent to the decimal number 20. You can change color 6 to red, or purple, or whatever you like, by storing a different number in the color register. This is done with **SetPalette** in Turbo Pascal and similar procedures in other languages.

Even that isn't powerful enough for the VGA's 262,144 colors. On the VGA, each color is defined by three intensities—red, green, and blue—each on a scale of 0 to 63. These are stored in a set of 256 *video DAC registers,* normally used in blocks of 64. Values can be stored in DAC registers with Turbo Pascal's **SetRgbPalette**.

In 320 x 200 256-color mode, the color code for each pixel refers to a DAC register directly. In the other graphics and text modes, there are still only 16 colors usable at a time, and a mapping scheme like that of the EGA comes into play. Color numbers map onto 16 EGA-like color registers, and these in turn contain not color definitions, but pointers to the DAC registers (Figure 4).

So if you store, say, 57 in color register 8, you're no longer defining color 8 to be rgbRGB 111001 (decimal 57); instead, you're defining color 8 as whatever is in DAC register 57. Fortunately for EGA aficionados, the default DAC register values match the EGA's rgbRGB codes rather closely.

Herein lies a trap for the unwary. Colors 0 to 15 do *not*, by default, map onto DAC registers 0 to 15. You can easily make them do so with **SetPalette**, and I recommend doing this if you are going to work with DAC registers. It makes life simpler.

Analog video—*DAC* stands for *digital-to-analog converter.* The CGA and EGA transmit color definitions to the monitor digitally, in IRGB or rgbRGB format, but the VGA outputs *analog video*—a voltage level that indicates the brightness of each color on a continuous scale. This analog signal comes from the DAC.

Whereas a digital signal merely has to be recognizable as *yes* or *no*, an analog signal has to be transmitted with no change at all. Even the slightest distortion will affect the picture.

This has important consequences. Many a $700 monitor displays a blurred picture because of a $15 cable. Analog video has to travel through impedance-matched 75-ohm coaxial cables to avoid degradation. IBM's monitor cables have miniature coaxial cables inside them, but some competitors' products don't. Beware the cheap VGA extension cord.

The mixer program—So if you have 262,144 colors, how do you decide which ones to use? That's the purpose of the color-mixing program (Listing 1). It displays three colors on the screen, including examples of all combinations of these colors as text and background. You pick a color to edit and then alter its hue, intensity, or saturation.

The program uses a number of tricks to keep the code short. It has its own version of **SetRgbPalette**. (Turbo Pascal's doesn't work in text mode—an unfortunate oversight.) The four big color patches are displayed in colors 1, 2, 3, and 4, which map onto DAC registers 1, 2, 3, and 4 respectively, and only the DAC registers are altered during execution.

The main loop accepts keystrokes and stores them in the string **Keys**. This string normally has zero or one elements—so why is it a string? Because upon receiving a PgUp or PgDn, the program stuffs five up or down arrows, respectively, into **Keys** and then lets the up and down arrow routines process them. Devious but concise.

How colors are manipulated—For each color, you can alter the red, green, or blue levels, intensity, or saturation. Altering the red, green, or blue level in the color definition is simple enough. Altering intensity is almost as simple—just increase or decrease all three colors in unison.

Here, an interesting problem arises. Suppose a color starts out as, say, red-green-blue 20-22-33, and you reduce the intensity to a tenth of its original value. The best you can do is use 2-2-3, which doesn't quite preserve the ratios between the numbers. Yet if you then raise the intensity, you'd like it to come back up to 20-22-33, not 20-20-30.

For this reason, the color values are stored internally in floating-point. You can't really reduce 20-22-33 to 2-2-3; instead, it will be 2.0-2.2-3.3, which rounds to 2-2-3 for copying into the DAC register but preserves the original ratios so that the original color can be recovered by multiplying the values by equal factors.

Changing saturation is even trickier. The basic idea is to increase saturation by increasing the strongest color and decreasing the weaker ones. If one of the color values is already 0, saturation cannot be increased. Otherwise, the highest color value (called **Top**) remains the same while the other two are decreased in proportion to how far away from **Top** they already are. This isn't a perfect way to adjust saturation, but it's convenient and subjectively satisfying. Decreasing saturation works the same way, in the opposite direction.

Using what you've made—Finally, here are some hints for using screen colors effectively.

1. Choose colors with lower saturation. Screens don't have to be gaudy. A silver-on-vellum effect (yellowish gray on maroon) is pleasant for full screen editing. Black, brown, or maroon on gray also works well.

2. Never use red type on a blue or green background, or vice versa. The human eye cannott focus red and blue together; a blurred effect results, no matter how sharp the screen is.

3. Remember that some VGAs have monochrome monitors; if you want two colors to contrast, make them differ in intensity as well as hue. To preview your VGA graphics in monochrome mode on a color monitor, execute this code:

```
VAR r: Registers;
...
r.ah := $10;
r.al := $1B;
r.bx := 0;
r.cx := 255;
Intr($10,r);
```

This reprograms all the DAC registers with the gray-scale equivalents of the colors they previously contained.

Michael Covington does artificial intelligence research at the University of Georgia, is the author of Astrophotography for the Amateur, *and is the co-author of* Dictionary of Computer Terms *and* Prolog Programming in Depth.

▲ FIGURE 4—The VGA Color Registers

On the VGA, the 16 color registers contain pointers to the 256 DAC registers (used in blocks of 64). Default colors and color register contents are the same as on EGA.

Color number	Color register	DAC register address	R	G	B	
0	0	0:	0	0	0	black
1	1	1:	0	0	42	blue
2	2	2:	0	42	0	green
3	3	3:	0	42	42	cyan
4	4	4:	42	0	0	red
5	5	5:	42	0	42	magenta
6	20	...				
7	7	7:	42	42	42	gray
8	56	...				
9	57	20:	42	21	0	brown
10	58	...				
11	59	56:	21	21	21	dark gray
12	60	57:	21	21	63	light blue
13	61	58:	21	63	21	light green
14	62	59:	21	63	63	light cyan
15	63	60:	63	21	21	light red
		61:	63	21	63	light magenta
		62:	63	63	21	yellow
		63:	63	63	63	white

User-modifiable   Also user-modifiable

▲ LISTING 1—VGAMIX.PAS

```
PROGRAM VgaColorMixer;
{ Michael A. Covington 1990 }

USES Crt,Dos;

CONST  Quality: ARRAY[1..5] OF String[12] =
       ('Redness','Greenness','Blueness','Saturation','Intensity');
CONST
  C: INTEGER = 1;    { Color being edited   }
  Q: INTEGER = 1;    { Quality being edited  }
  R: ARRAY[1..3] OF REAL = (63,  0,  0);   { Red component }
  G: ARRAY[1..3] OF REAL = ( 0, 63,  0);   { Green component }
  B: ARRAY[1..3] OF REAL = ( 0,  0, 63);   { Blue component }

PROCEDURE  SetRgbPalette(ColorNum,Red,Green,Blue:INTEGER);
{ Like the SetRgbPalette procedure provided in GRAPH.TPU, but
does not require .BGI files. Copy and use in your own programs. }
VAR
   R: Registers;
BEGIN
   R.ax := $1010;
   R.bx := ColorNum;
   R.dh := Red;
   R.ch := Green;
   R.cl := Blue;
   Intr($10,R)
END;

PROCEDURE  HideCursor;
{ For VGA and most others. Undone by textmode(co80). }
VAR
   R: Registers;
BEGIN
   R.cx := $2000;  { Start cursor on scan line $20, end on $00 }
   R.ah := 1;      { That is, end it before it starts }
   Intr($10,R)
END;

PROCEDURE  Block(Left,Upper,Right,Lower,Color:  INTEGER);
VAR
  Row, Col: INTEGER;
BEGIN
  TextColor(Color);
  FOR Row := Upper TO Lower DO
    FOR Col := Left TO Right DO
      BEGIN
        GoToXY(Col,Row); write(#219)
      END;
  TextColor(White);
END;

PROCEDURE  Box(Left,Upper,Right,Lower,Color:  INTEGER);
BEGIN
   Block(Left,Upper,Left,Lower,Color);
   Block(Right,Upper,Right,Lower,Color);
   Block(Left,Upper,Right,Upper,Color);
   Block(Left,Lower,Right,Lower,Color)
END;

PROCEDURE  WriteCentered(Msg:String;Row,Color:INTEGER);
BEGIN
  GoToXY(40-(length(Msg) div 2),Row);
  write(Msg)
END;

PROCEDURE  WriteInverse(Msg:String);
BEGIN
  TextBackground(White);
  TextColor(Black);
  write(Msg);
  TextColor(White);
  TextBackground(Black)
END;

PROCEDURE UpdateColors;
{ Updates just those parts of the screen that change }
{ when the user alters a color quality }
VAR
  j, red, green, blue: INTEGER;

BEGIN
  SetRgbPalette(4,round(R[C]),round(G[C]),round(B[C]));
    { Color 4 will always be the color currently being edited }
  FOR j:=1 TO 3 DO
    BEGIN
      SetRgbPalette(j,round(R[j]),round(G[j]),round(B[j]));
      { Label the colors }
      TextColor(White);
      GoToXY(20*j-3,9);
      IF j=C THEN
        WriteInverse('Color '+chr(ord('0')+j))
      ELSE
        write('Color '+chr(ord('0')+j));
      GoToXY(20*j-7,7);
      IF j=C THEN
        TextColor(White)
      ELSE
        TextColor(LightGray);
      Write(  'R=',round(R[j]):2,
            '  G=',round(G[j]):2,
            '  B=',round(B[j]):2);
    END;
  { Update the menu of qualities }
  TextBackground(Black); TextColor(White);
  GoToXY(11,19);
  FOR j:=1 TO 5 DO
    BEGIN
      IF j=Q THEN
        WriteInverse(Quality[j])
      ELSE
        Write(Quality[j]);
      Write('     ')
    END
END;

PROCEDURE  UpdateScreen;
VAR
  j,k: INTEGER;
BEGIN
  TextMode(Co80); { Clears screen and resets colors }
  HideCursor;
  UpdateColors;
  Box(1,1,80,21,DarkGray);
  WriteCentered('V G A   C o l o r   M i x e r',3,White);
  WriteCentered('TAB chooses color to edit',22,White);
  WriteCentered( #$1B + ' ' + #$1A + ' choose a quality to alter',
        23,White);
  WriteCentered( #$18 + ' increases and ' + #$19 + ' decreases that
             quality', 24,White);
  WriteCentered('Alt-X ends program',25,White);
  { Color swatches }
  Block(11,5,29,6,1);
  Block(31,5,49,6,2);
  Block(51,5,69,6,3);
  { Large patch of the color currently being edited }
  Block(11,11,69,15,4);
  { Text samples }
  GoToXY(10,17);
  FOR j:=1 to 3 DO
    FOR k:=1 TO 3 DO
      IF j<>k THEN
        BEGIN
          TextBackground(Black); Write(' ');
          TextBackground(j);
          TextColor(k);
          Write(' ',k,' on ',j,' ')
        END;
  TextBackground(Black);
END;
```

```
FUNCTION Min(X,Y,Z:REAL):REAL;
BEGIN
  IF X<Y THEN
    { Minimum is not Y }
    IF X<Z THEN Min:=X ELSE Min:=Z
  ELSE
    { Minimum is not X }
    IF Y<Z THEN Min:=Y ELSE Min:=Z
END;

FUNCTION  Max(X,Y,Z:REAL):REAL;
BEGIN
  IF X>Y THEN
    { Maximum is not Y }
    IF X>Z THEN Max:=X ELSE Max:=Z
  ELSE
    { Maximum is not X }
    IF Y>Z THEN Max:=Y ELSE Max:=Z
END;

VAR   { Main }
  Keys: string;
  Top, Factor: real;

BEGIN
 UpdateScreen;   Keys := '';
 WHILE TRUE DO
 BEGIN
  IF Keys = '' then Keys := ReadKey;
  CASE Keys[1] OF
    #09 : { Tab }
           BEGIN
             C := C MOD 3 + 1;
             UpdateColors
           END;
    #27 : { First byte of any non-ASCII key } { do nothing };
    #72 : { Up arrow }
           BEGIN
             CASE Q OF
               1: IF R[C]<62.5 THEN R[C] := R[C]+1;
               2: IF G[C]<62.5 THEN G[C] := G[C]+1;
               3: IF B[C]<62.5 THEN B[C] := B[C]+1;
               4: BEGIN { Up saturation }
                    Top  := Max(R[C],G[C],B[C]);
                    IF Min(R[C],G[C],B[C]) > 0.5 THEN
                    BEGIN
                      Factor := 1/Abs(Top-Min(R[C],G[C],B[C]));
                      R[C] := R[C] + Factor*(R[C] - Top);
                      G[C] := G[C] + Factor*(G[C] - Top);
                      B[C] := B[C] + Factor*(B[C] - Top)
                    END
                  END;
               5: { Up intensity  }
                    IF Max(R[C],G[C],B[C])<62.5 THEN
                    BEGIN
                      R[C] := R[C]*1.01;
                      G[C] := G[C]*1.01;
                      B[C] := B[C]*1.01
                    END
             END;
             UpdateColors
           END;
    #73 : { PgUp - five up arrows }
           Keys := Keys[1]+#72+#72+#72+#72+#72+copy(Keys,2,255);
    #80 : { Down arrow }
           BEGIN
             CASE Q OF
               1: IF R[C]>=0.5 THEN R[C] := R[C]-1;
               2: IF G[C]>=0.5 THEN G[C] := G[C]-1;
               3: IF B[C]>=0.5 THEN B[C] := B[C]-1;
               4: { Down saturation }
                  BEGIN
                    Top  := Max(R[C],G[C],B[C]);
                    IF (Top-Min(R[C],G[C],B[C])) > 0.5 THEN
                    BEGIN
                      Factor := 1/Abs(Top-Min(R[C],G[C],B[C]));
                      R[C] := R[C] - Factor*(R[C] - Top);
                      G[C] := G[C] - Factor*(G[C] - Top);
                      B[C] := B[C] - Factor*(B[C] - Top)
                    END
                  END;
               5: { Down intensity  }
                  BEGIN
                    R[C]:=R[C]*0.99;
                    G[C]:=G[C]*0.99;
                    B[C]:=B[C]*0.99
                  END
             END;
             UpdateColors
           END;
    #81 : { PgDn - five down arrows }
           Keys := Keys[1]+#80+#80+#80+#80+#80+copy(Keys,2,255);
    #75 : { Left arrow }
           BEGIN
             IF Q > 1 THEN Dec(Q);
             UpdateColors
           END;
    #77 :   BEGIN { Right arrow }
             IF Q < 5 THEN Inc(Q);
             UpdateColors
           END;
    #45 :  BEGIN  { Alt-X }
             TextMode(Co80); { Reset colors }
             Halt
           END
  END {Case};
 Delete(Keys,1,1); { Eat the keystroke that was just acted on }
 END
END.
```


Circle 45 on reader service card