COMPUTER GRAPHICS AND IMAGE PROCESSING 10, 297–317 (1979)

Color Map Techniques

KENNETH R. SLOAN, JR., AND CHRISTOPHER M. BROWN

Computer Science Department, University of Rochester, Rochester, New York 14627

Received July 31, 1978; revised February 9, 1979

Full color, raster-scan displays are increasingly popular media for computer graphics. Several bits of information about every picture element in a raster image can be stored in a local image memory and used to produce a video display. A color map is a look-up table interposed between the image memory and the video display generator which can be used to implement an arbitrary mapping from image-memory values to display values. We present several techniques and applications which depend upon such a color map to provide fast and flexible changes to the display without rewriting the image memory.

1. INTRODUCTION

Full color, raster-scan displays are increasingly popular media for computer graphics. As mass memories drop in price, it has become economically feasible to store and display color images at resolutions comparable to that of a standard television set (Negroponte [11]). Typically, the image is represented as a complete array of picture elements (pixels), with each pixel representing a small area of fixed size and position in the image. A pixel, in turn, is represented as a set of bits, usually organized as a set of single-bit "planes." A binary image requires a single-bit plane, which may be interpreted as "Black if 1, White if 0." Where several bits are used, they may be interpreted naturally as a gray-scale intensity or as a set of Red, Green, and Blue intensities. In simple hardware configurations, the image-memory planes have fixed interpretations; for example, 12 planes may always be interpreted as Red, Green, and Blue intensities of 4 bits (16 levels) each. The interpretation is done by a video display generator which continuously scans the image memory and produces a video signal which is displayed on a monitor (binary, gray-scale, or color). If the selection of bit planes going to the digital-to-analog converters (DACs) of the video generator is hard-wired, the configuration is sometimes called "true color." Color maps provide a method of allocating bit planes flexibly; they include the true color configurations as a special case.

Fixed interpretation of the image planes (a true color configuration) is satisfactory for many applications; however, when the images being processed (or created) do not have a "natural" color or intensity, this fixed display mode is unnecessarily restrictive. The pixels may have interpretations such as tempera-

0146-664X/79/080297-21$02.00/0

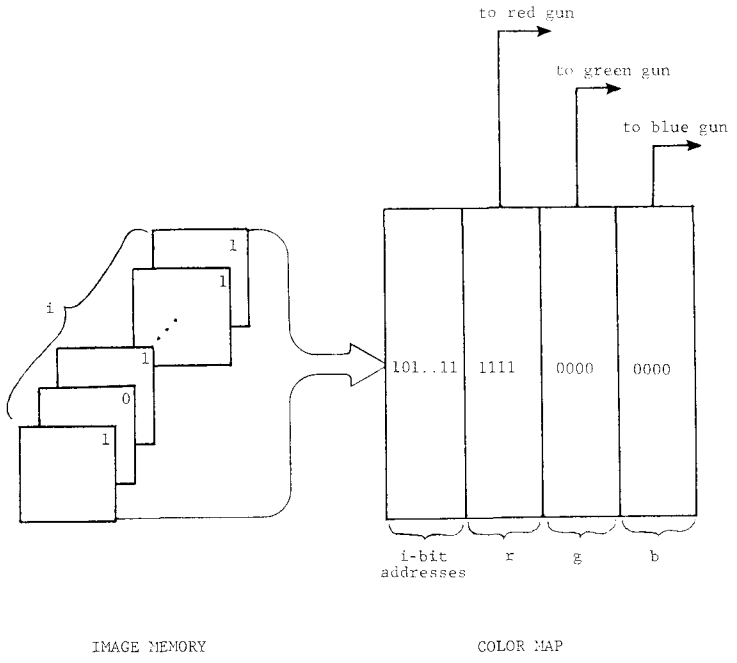

FIG. 1. The color map. An i-bit pixel in the image memory is used as an index into the color map. At this address in the map is found a value which is interpreted by the hardware as a color or intensity for display.

ture, or population density; it is often useful to display such data by using color to accentuate the interpretations, calling attention to phenomena of interest. There may be several interesting phenomena, however, and to create a different emphasis might mean recoloring the entire image. Indeed, for many colored images, quite often it is desirable to fix the form of the image and manipulate its colors. With a fixed display interpretation, changing the colors necessitates rewriting the entire image; the colors or intensities of pixels must be recalculated and rewritten into the image memory to provide a different-looking display of essentially the same data.

To reduce such wasteful data shuffling, a so-called "color map" can be interposed between the image memory and the display. The color map is a table specifying a transformation between the image-memory bits and the bits presented to the display (Fig. 1). The table is addressed by the image-memory pixel values. The contents of the table at each address is a field of bits whose width is unrelated to the image-memory pixel size; it is this table contents, not the pixel value, which determines which signals are sent to the monitor.

Thus, if the image memory consists of a single-bit plane, there are just two addresses in the color map, one for each pixel value. The color map values at these addresses may be quite large and have multiple subfields; they may be 24 bits wide, say, specifying 8 bits of intensity to be sent to each of three color guns in a color monitor. If the single-bit image memory above is 1024 pixels

square, then the colors appearing on the display may be changed by changing the two values in the color map rather than the million pixels of image memory.

The color map mechanism is an extremely useful tool, with many interesting applications. Its original justification lies in applications involving the "false coloring" of essentially gray-scale data. The purpose of this paper is to present a collection of raster-graphics techniques and applications which depend upon the color map for their effectiveness. It is seen that this relatively simple hardware device can greatly extend the usefulness of raster-graphics devices. Many of these techniques are "classical," or at least have been independently conceived by several other workers; in this paper we only desire to contribute to the growing awareness of potentialities inherent in current raster-graphics hardware.

2. HARDWARE ASSUMPTIONS

The cost effectiveness of the color map in the techniques presented below depends to a certain extent upon the actual hardware configuration involved. We assume that the available hardware consists of:

(a) Gray-scale monitor	h bits
(b) RGB monitor	$r + g + b$ bits
(c) Image memory	$x*y*i$ bits, with $x*y \gg 2**i$
(d) Color map	$i \rightarrow h$ and $i \rightarrow (r, g, b)$.

The image memory consists of i planes, each an x by y array of bits. Pixels i bits deep are located in the x by y raster.

Assumption (c) is a key one. It means that it will be much faster to rewrite the color map than to rewrite the image memory.

The gray-scale monitor displays an image derived by treating h bits as an integer intensity value. The RGB monitor treats r, g, and b bits (disjoint) as intensity values for Red, Green, and Blue components of the image. If identical levels are always sent to each color gun of an RGB monitor, a gray-scale image results. The restriction of identical components means that $\min(r, g, b)$ bits of gray scale may be displayed. Any other device expecting some number of signal levels can be accommodated with subfields of the map contents, as long as the map output is wired accordingly.

One commercially available and relatively modest configuration, used occasionally below for examples, consists of a 512*256*12-bit image memory, an 8-bit DAC for the gray-scale video monitor, three 4-bit DACs for the RGB video monitor, and a $12 \rightarrow 12$-bit color map.

3. APPLICATIONS

The following sections present specific applications of the color map facility. The key idea behind all of them is that the color map provides a simple method of restructuring the image-memory planes without requiring excessive computation or hardware.

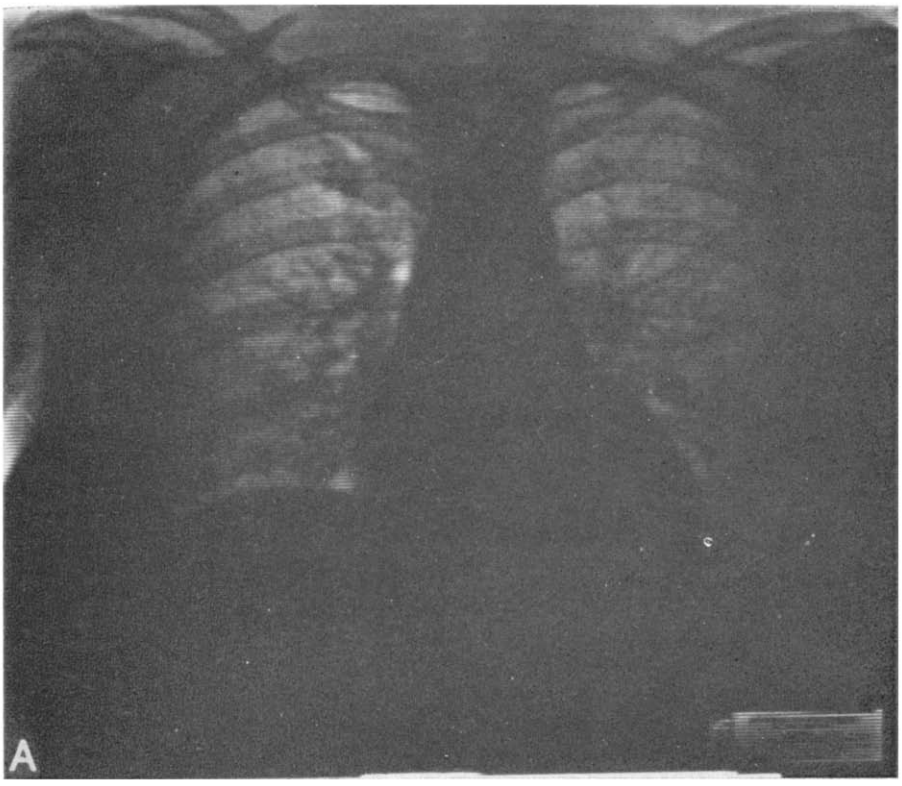

FIG. 2. Figure (A) Pixel values 0–4095 scaled linearly (gamma = 1) between output intensities 0 and 255. (B) Linear scaling takes place across the range of interesting pixel values, 125–800. (C) Same range as (B) but gamma = 0.5.

3.1. Intensity Transformation

In a gray-scale image, or within the colors of an image consisting of color separations, it is sometimes advantageous to send to the display not an intensity which is linearly related to the pixel contents, as would be provided by a DAC working on the pixel, but an intensity related by another function.

Common general schemes include "gamma correction," logarithmic transformation, and various histogram modification techniques. Gamma correction and logarithmic transformation of the image intensities are monotonic, and compensate for nonlinearities in the display device and the human visual system (Blinn [3]). Some digitizers and displays provide a piecewise-linear "gamma correction" function which has a small number of variable breakpoints to perform these functions in hardware. Histogram modification is usually more complicated and is commonly performed on the digitized image itself (Hummel [7]).

All of these intensity transformations can be done very easily with a writable color map under program control. In this application, if intensity transformation

$$\text{Displayed Value} = \text{Function (Image-Memory Value)}$$

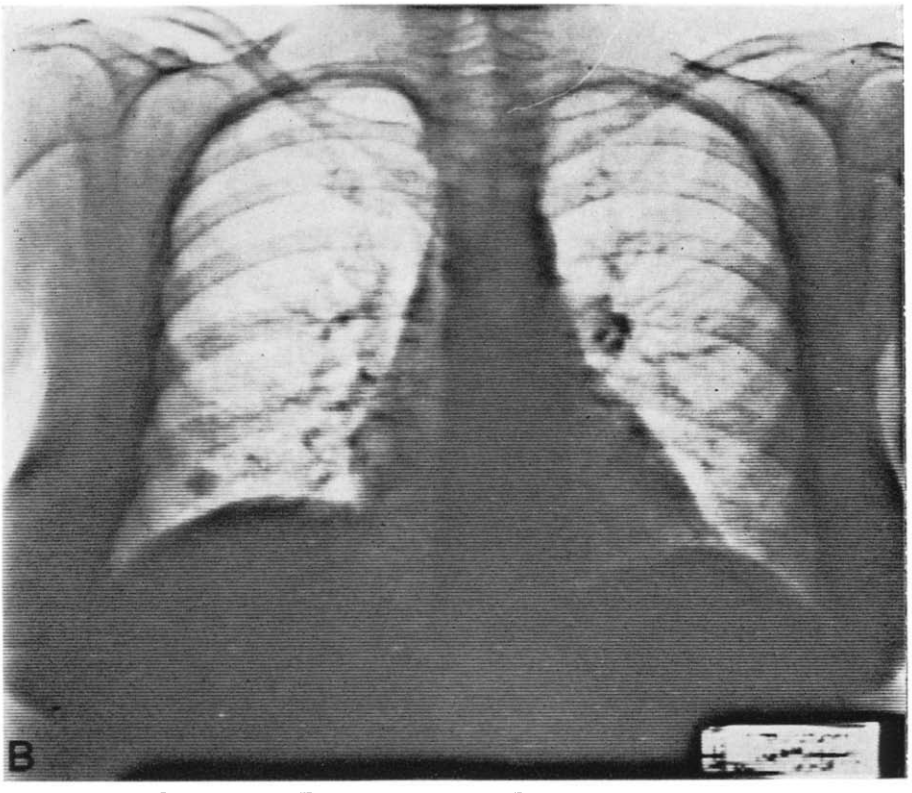

FIG. 2—*Continued.*

is to be implemented, the color map at address Image-Memory Value is simply given the value Function(Image-Memory Value). The parameters of such a function can be interactively manipulated by the viewer, without rewriting the image memory.

For example, Fig. 2 shows a chest radiograph, digitized to 12 bits, Fourier filtered, and displayed on an 8-bit (256 gray level) monitor. The image is stored in a 12-bit image memory. The full range of pixel values is [0,1820], with the "interesting" pixels having a range of approximately [150, 750]. Calculation of the intensity transformations shown involves three parameters: min, max, and gamma. Min and max specify the range of pixel values which the viewer considers "interesting." Over this range of Image-Memory Values,

$$\text{Displayed Value} = (\text{Image-Memory Value})^{**}(1/\text{gamma}).$$

The Displayed Values are then linearly scaled to cover the range [0, 255], and Image-Memory Values greater than (less than) max (min) are mapped to 255 (0). Thus, a slice of Image-Memory Values may be selected, "gamma corrected," and spread over the available range of the monitor. (See Fig. 2.)

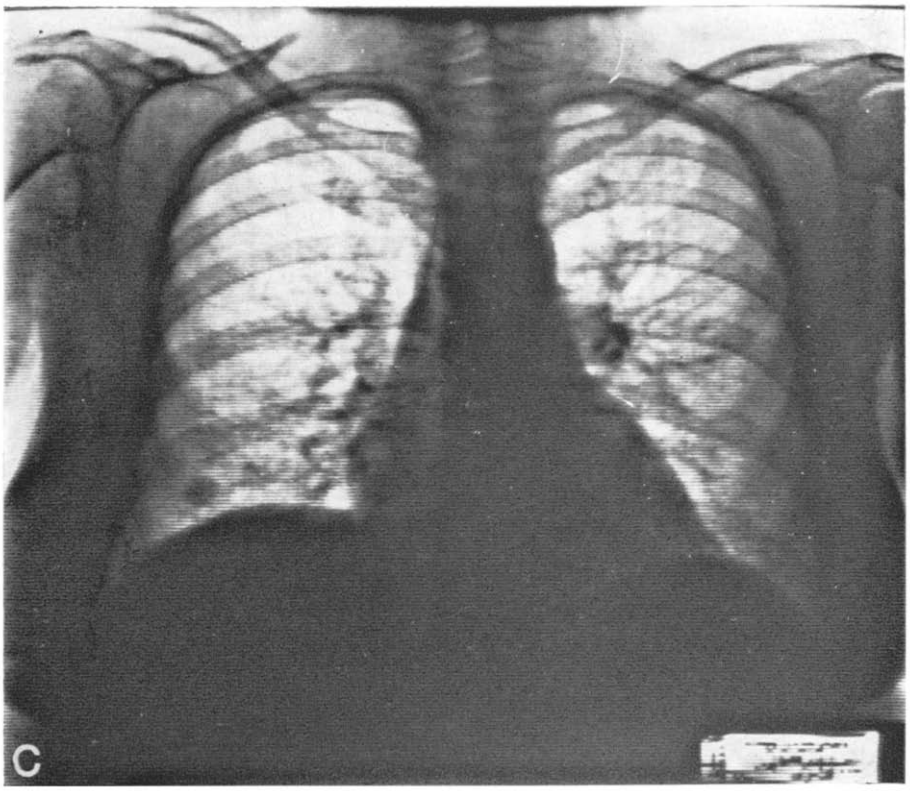

FIG. 2—*Continued.*

3.2. False Color (Gray Scale → RGB)

A related application of the color map is to provide a "false color" or "pseudo-color" display of a gray-scale or nonintensity image. Even if a black- and-white monitor could effectively display 256 levels of gray (there is general agreement that 64 gray levels are all one can expect), the "perceptual distance" between such gray levels is very small; that is, it is hard to tell the difference between them. On the other hand, it is easy to find 256 colors which can be reliably differentiated; there is more perceptual distance to be had in color space. Thus converting an image to color is a popular way of bringing out subtle differences in intensity. The addition of color to the display allows the observer to discriminate between adjacent regions which are only slightly different in gray scale, as well as identify regions which are spatially separated but identical in intensity. If one wants to preserve the idea of ordering the pixel values along some one-dimensional scale, it is necessary in false coloring to select a set of colors with an easily remembered order (so that the relative intensities can still be determined). To be most effective, the colors should be maximally distinguishable by the observer. Thus the original pixel intensities are treated as the arc-length parameters in a parameterized path through color space. Different paths

through color space may be used, depending upon the specific criteria, including task domain, human perception, and display characteristics (Badler and Miller [1]; Booth and Schroeder [4]). Due to psychophysical phenomena and hardware characteristics, it is not always easy to do seemingly simple things, such as mapping intensities onto a "spectrum" of red to blue.

The major advantage of using a hardware color map for this "false coloring" is flexibility. The essence of false coloring is the transformation of gray-scale (or one-dimensionally ordered) data into color data, which is usually conceived of as being three-dimensional, or characterized by three values. It may be useful to try several different false colorings (perhaps representing different thresholds, or different paths through color space). The color map encourages such experimentation, allows colorings to be quickly changed for different emphases, and, more important, preserves the original data in the image memory.

Looking back to the assumed hardware configuration, we see that it is possible to store a gray-scale image, or nonintensity pixel data, with i bits of precision. The image may be displayed on the gray-scale monitor with h bits of precision. Now, merely by rewriting the color map, we can view the image on the color monitor as a false color image. Changing the false color mapping or returning to the original image may all be done by computing and writing $2**i$ entries in the color map, rather than the $x*y$ entries in the image memory.

3.3. Overlays

The contents of the image memory need not be considered as a single, atomic image. Complex images may be constructed by overlaying several independent subimages. For example, the production of an animated film may involve the superposition of several image planes, each of which can be manipulated independently. Foreground figures can move about, obscuring a background image without destroying it. Overlays of colored lines are useful for pointing out or delineating structures in a gray-scale image. Overlays are often implemented by special-purpose hardware which enforces priorities among the various overlay planes and the image memory, thus specifying the order in which planes overlay each other. It is possible to design color mappings which not only implement any desired assignment of planes as overlays of arbitrary color, but which allow various options as to both the priority ordering and transparency or opacity of the overlays. Priorities among the overlay planes may be changed dynamically.

In order to implement an overlay scheme, there must be a set of encoding and decoding functions which allow each i-bit image-memory pixel to represent the corresponding pixels in each overlay. This may be done by assigning specific bit planes in the image memory to specific overlays, but this is not absolutely necessary. Specific bit plane assignments are especially useful when the image memory can be accessed or modified on a bit plane basis. In any event, the knowledge of the encoding function makes it possible to design color mappings which selectively display the various overlays.

For example, consider the problem of displaying map-like information in conjunction with a corresponding gray-scale aerial image. We would like to see

the original gray-scale image, with linear or area feature overlays. For clarity, the overlays should be in color, to distinguish the several overlays from each other and from the gray-scale data beneath. In addition, we want to control the priorities of the overlays, the appearance of the original data, and the colors of the overlays. Of course, we do not want to erase or rewrite the image memory to exercise these display options.

One solution to this problem, in use at our laboratory, is to allocate h bits of the image memory for the gray-scale image, leaving $(i - h)$ bits for the overlay information. Each of these $(i - h)$ overlay planes is treated as an independent binary image. Thus, we have $(i - h) + 1$ independent images.

Once these images have been written into the image memory, we can control the display by manipulating the color map. We can now chose to display the complete gray-scale image on the gray-scale monitor (with full precision), a reduced precision ($\min(r, q, b)$ bits) gray-scale image on the RGB monitor with any combination of the colored binary overlay images, or just the binary overlays against a constant color background. The binary overlays may be assigned arbitrary colors. They may be made invisible, act as colored filters for what lies beneath, or be opaque. The order in which they are stacked with respect to the viewer may also be manipulated. The effect is as though the gray-scale image and the overlays were transparencies on an overhead projector, perfectly registered, and of variable color, density, and order.

To exercise these options, the user must specify:

(a) the monitor (Gray-scale vs RGB),
(b) a permutation (a priority ordering) of the binary overlays,
(c) a color for each binary overlay,
(d) a density for each binary overlay (on a scale from opaque to invisible), and
(e) how to use the gray-scale image (either beneath the binary overlays in one of a number of forms, or replaced by a constant background).

From this information, it is relatively simple to compute the proper values of the color map entries to be interpreted by the hardware as red, green, blue, and/or gray-scale bits. A value must be computed for each possible combination of the $(i - h)$ overlay bits and h image bits possible as values in the image memory.

Figure 3 shows a small example involving a 3-bit image memory, a color monitor with 1-bit input to each of its red, green, and blue color guns, and a black-and-white monitor with 2-bit gray-scale input taken from the low-order 2 bits of the color map values. The example scales up easily to larger configurations with more overlays. In this example, the high-order (red) bit of each pixel is chosen as an overlay bit. The remaining two low-order bits are used to define the gray-scale image. The problems are how to implement the overlay, and how to compress the 2-bit gray-scale image for display in black and white on the color monitor. The solution to the latter problem is to replicate the high-order gray-scale intensity bit and send it to all three color guns; this effectively reduces the gray-scale resolution from four intensities to two intensities, as expected. Mixing in the red overlay is accomplished by rewriting the map in an appropriate way depend-

Pixel value			Map 1 (Two-bit gray scale)			Map 2 (Invisible)			Map 3 (Opaque)			Map 4 (Tinted)		
			B & W			R	G	B	R	G	B	R	G	B
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0	0	0	0	0	0	0	0
0	1	0	0	1	0	1	1	1	1	1	1	1	1	1
0	1	1	0	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	1	0	0	1	0	0	0	1	0	0	0	0	0
1	1	0	0	1	0	1	1	1	1	0	0	1	0	1
1	1	1	0	1	1	1	1	1	1	0	0	1	0	1

FIG. 3. Four color maps for three-bit pixel values. The high-order bit in the pixel has the interpretation of a red overlay. The two low-order bits contain gray-scale information. In the two-bit gray-scale map, the overlay is ignored and the two low-order gray-scale values are sent without transformation to the black-and-white monitor. The remaining three maps are for the color monitor. In the first, the overlay is invisible, and a black-and-white image results from sending the same information to all three color guns in parallel. This reduces the gray-scale resolution; in fact, the image on the color monitor is binary, not four-valued. In the third map, whenever the overlay bit is on, a pure red is sent to the display. In the fourth map, the red overlay is acting as a filter for image intensity. When there is no intensity (pixel values 100 and 101), the overlay does not appear. When there is intensity (110, 111), this map specifies a purplish color. In the last two maps, when the overlay bit is off, the binary image of the second map is seen. The choice of particular colors for display when multiple tinted overlay are superimposed may be left to the judgment of the color map designer or derived from a model of the color absorption of the tinted overlays.

ing on what the overlay is to look like. Figure 3 shows four maps, implementing 2-bit gray-scale images for the black-and-white monitor, and invisible, opaque, and tinted overlays on a reduced gray-scale image for the color monitor.

In our example configuration, to display an 8-bit gray-scale image on a $4 + 4 + 4$-bit RGB monitor, the color map values are based on the 4 most significant bits in the image; these 4 bits are sent in parallel to the three color guns to produce a 16-valued gray-scale image on the RGB monitor. The remaining 4 high-order bit planes of the 12-bit image memory are used to specify four binary overlays. If a particular overlay is to be shown, its color and density are mixed with the other overlays and the gray-scale image to produce the overlay effect. This calculation must be done for each of the 4096 possible combinations. The exact nature of the color mixtures (where several overlays overlap) and the manner in which the overlays obscure the gray-scale data are matters of individual taste. With a small number of overlay colors, all of the possible mixtures can be anticipated and specified explicitly. This is the approach we have taken in our implementation, in which the individual overlay colors are restricted to the set (R, G, B, Y). When the set of initial colors is more flexible, the mixing could be based upon a general color model.

In this example, the choices concerning the number of bits allocated to the gray-scale image and the overlays (8 bits and 1 bit each) were motivated primarily

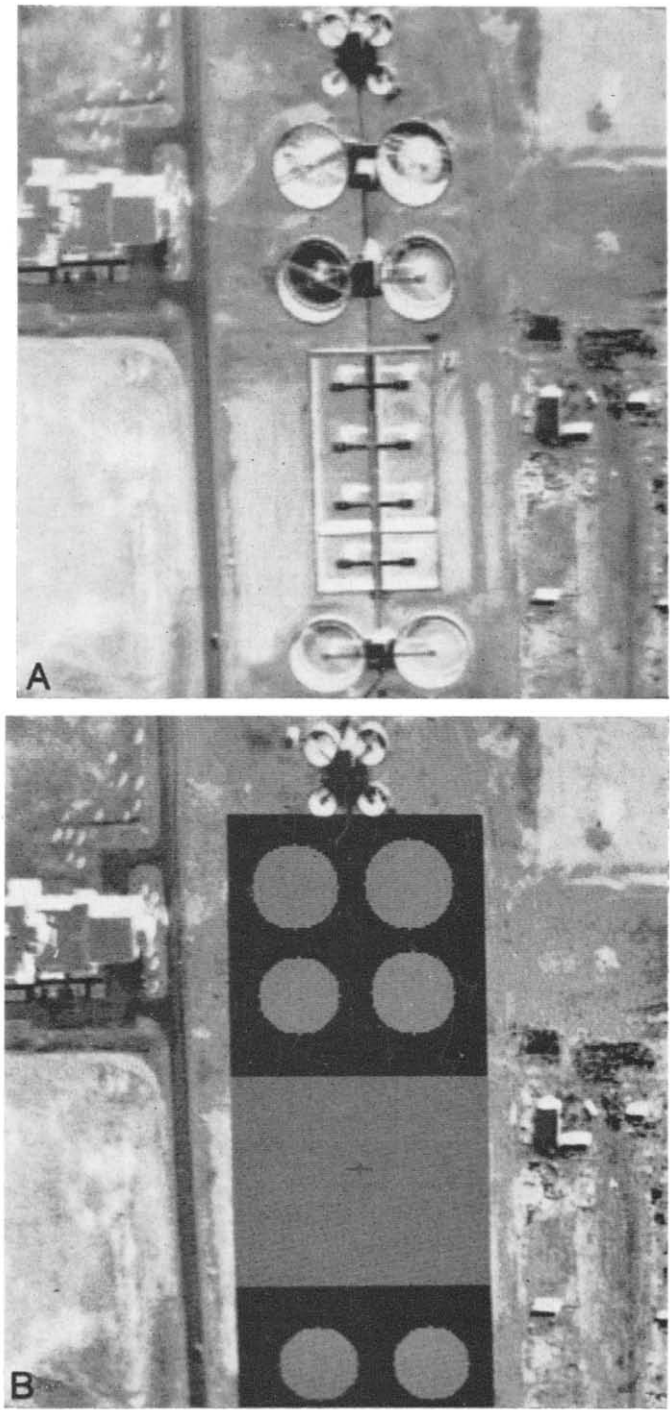

FIG. 4. The image of (A) can be overlaid with various colors (here shown opaque for effect), as in (B).

by the characteristics of the hardware (an 8-bit gray-scale monitor) and by the expected nature of the overlay information (binary). Of course other configurations (such as nonbinary overlays) are possible. Another generalization is not just to use the high-order gray-scale bits to derive what should appear in reduced gray-scale on the color monitor. The overlay effect can be combined with intensity transformations such as gamma correction or histogram modifications. Examples of opaque overlays are shown in Fig. 4.

3.4. Motion

Full color, raster displays are capable of displaying images which are very complex in both spatial and color variations. The number of data required to specify a single frame of a color raster-TV image is quite large (e.g., 512*512*12 bits). The bandwidth needed in order to respecify such an image completely at a rate which gives the impression of smooth motion is usually so expensive that moving images are generated in non-real time by photographing the display one frame at a time, thus creating a film which is to be shown much later. Real-time motion graphics has traditionally been restricted to high-performance vector displays with transformation hardware.

When the frame-to-frame changes to the image are small, involving a relatively small subset of image pixels, there may be enough available bandwidth to "fix up" a raster image fast enough for flicker fusion to take place (a new frame approximately every 0.05 sec). The multiple-buffering technique mentioned below, done with the aid of the color map, can smooth the transitions between frames. When overlays, as discussed above, are used to provide static backgrounds, quite realistic motion of a small part of the total image may be produced.

Rewriting the image memory fast enough to provide motion effects is the basic challenge in motion raster graphics. This rewriting may be aided by using the color map to provide buffering and interframe smoothing services. First, the image-memory planes must be allocated among the several frames or overlays to be used. Second, the transitions between frames must be made as smooth as possible. Rapid writing to a restricted set of bit planes is aided by a standard choice of modes in a raster-graphics device. In one mode a pixel is addressed by its raster location and a fixed number of bits (usually i) is accepted; they specify the pixel value at that point in the raster, and i bit planes are changed. In the other mode, a location in the raster and a single plane are selected, and writing takes place along that plane, not through planes.

The image memory must be divided up into overlays if motion in separate overlays is to be allowed. The allocation of image memory is affected by the expected color complexity (how many different colors) and by the expected rate of change (changed pixels per frame) of each overlay. If an overlay is relatively stable, then many bits of intensity or color may be allocated to each pixel. On the other hand, if an overlay is liable to much change, fewer bits can be allocated to it, since many of its pixels may need to be changed in the available time. Once the image memory has been divided into overlays, the color map is used to combine the overlays into a single image and to expand the range of possible colors, as described in Section 3.3.

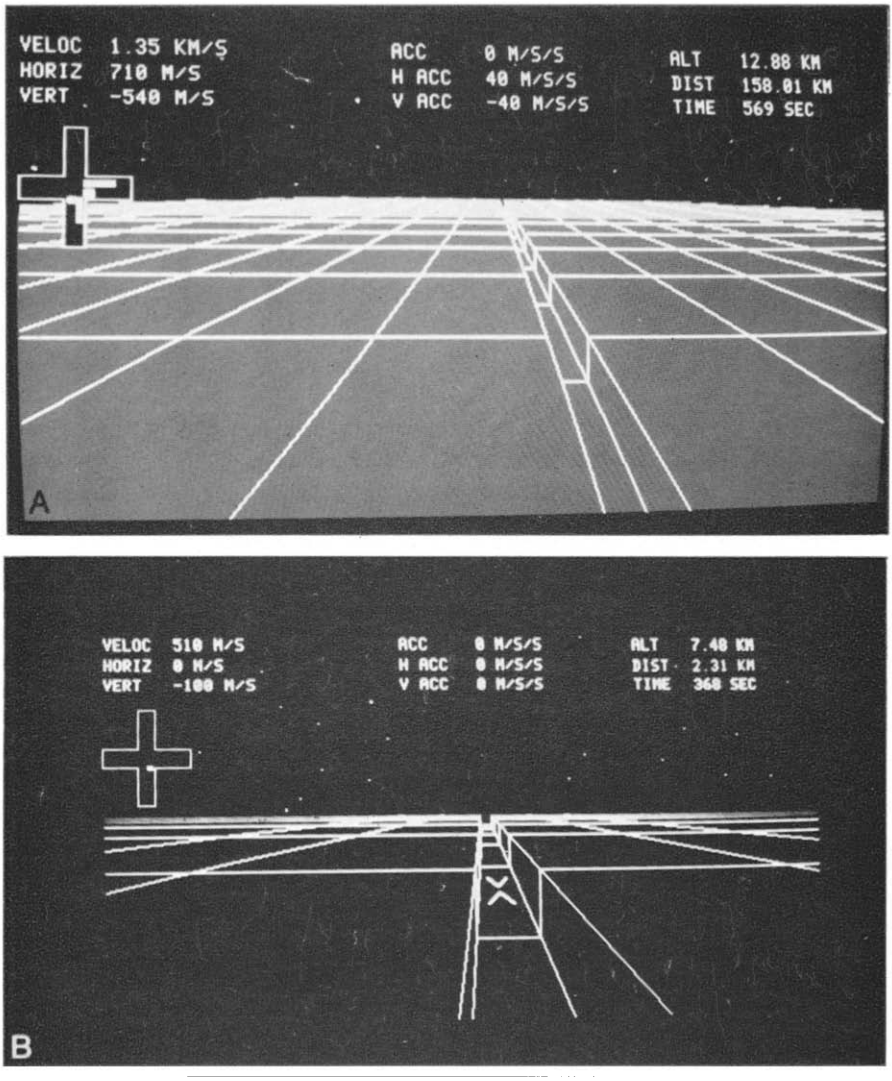

FIG. 5. Frames from DEATHSTAR, a dynamic interactive computer game. The image memory is used as a double buffer, each buffer holding four frames. With each buffer comes new information on the deathstar surface position, the star background, and color-coded cockpit instrumentation. The color map is not only used to accomplish the buffering, but used for every buffer, to run through four gridline overlays to show motion across the deathstar surface, and to provide varicolored enemy laser flashes, a moving target, shown in (B), and flashing varicolored explosion effects. In (C), the screen is flashing in color because of a laser hit; the laser flashes are beams emanating from the trench bottom.

The transitions between frames are guaranteed to be smooth if the image memory is implemented as a circular shift register, with both reads (by the display generator) and writes (the next-frame updates) synchronized. However, if most of the available access time is for refreshing, which is typically true, this

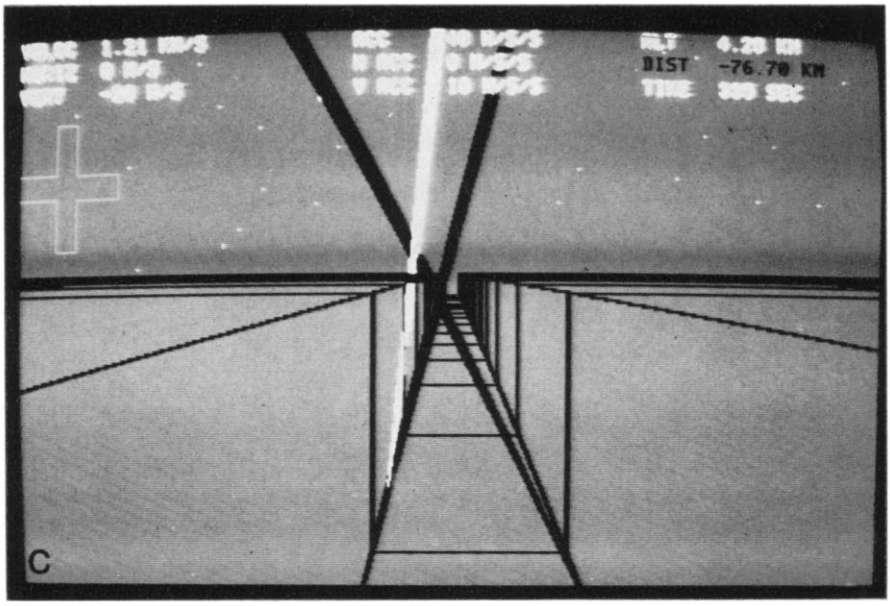

FIG. 5—*Continued.*

limits the number of updates which can be made. On the other hand, when the image memory is random-access and dual ported, the updating process can "beat" against the display generation process so that the image which is displayed contains half-completed updates. We can avoid this problem by further partitioning the image-memory planes into two halves. The color map is arranged so that, at any one time, half of the image-memory planes are ignored. The image memory contains two images (images or overlay sets): the current, displayed image, and the next image. When the next image has been completely written into the image memory, the color map is changed, switching quickly from the current image to the next image. In this way, we can "double-buffer" the display, using the color map to change buffers quickly. Of course, we pay a price for this in the reduced number of bits available to represent any single image. If f bits of color/intensity precision are needed for each frame, we can write i/f complete images in the image memory, using the color map to implement the sequencing. For very short, repetitive sequences, we can put an entire "loop" in the image memory (up to i binary frames). This technique could be called "false motion" if the image memory were never rewritten, and all motion came from selecting combinations of memory planes with the color map. Also, note that we can apply these motion techniques to a subset of the image-memory planes and show simple moving images over a complex, static background. By using less than the entire screen for the image, a speedup is obtained at the cost of reduced spatial resolution. One application of the technique of this section results in frames such as those shown in Fig. 5.

3.5. Using Intrinsic Property Images

The usual use of an image memory is to store one or more images by encoding color or intensity information for each pixel in the image memory. When interpreted by the video hardware in conjunction with some output device, the result is a color/intensity image. There is no a priori reason to associate with each image-memory pixel exclusively the visual information about how it looks; of interest also might be intrinsic information about the range or reflectivity of scene points corresponding to image-memory pixels (Barrow and Tenenbaum [2]; Nitzan *et al.* [12]). In the standard false-color situation, some intrinsic characteristic of a scene is interpreted as a visual color or intensity for the purposes of displaying the intrinsic property of interest.

An important and common method of producing 3-D shaded graphics involves going from scene properties (typically surface reflectance and orientation properties and lighting conditions) to intensity. The production of an image from a stored 3-D model involves accessing the model for relevant information and storing intensity information in the image memory according to the formula used to produce the shaded image. The idea naturally arises of storing relatively static intrinsic information in the image memory, and then implementing the shading calculation in the color map, so that it is done continuously in the hardware.

This idea is only satisfactory if there are enough bits of information in an image-memory pixel to store the needed static information for the shading calculation. Almost certainly surface orientation must be stored, and if this is to vary continuously, many significant bits may be necessary. The addition of color or reflectivity codes for pixels would use more bits. But if the information storable as an image-memory pixel is sufficient for the desired calculation, the entire scene may be "relighted" or "resurfaced" at the expense only of rewriting the color map; the calculation is done only once for every allowed combination of intrinsic qualities. This nonredundant calculation may be done and implemented by loading the color map in a short amount of time, leading to easy and efficient implementation of such effects as moving light sources, sources of changing color or brightness characteristics, and varying surface characteristics such as glossiness. The map makes possible quick experimentation with illumination conditions (such as light source placement), given a shading formula, and also quick experimentation with different shading formulas; the intrinsic image in the image memory is never changed nor directly displayed, as the entire transformation from surface properties to appearance is done via the color map.

For example, the image memory of the example configuration could be used for storing an intrinsic image consisting solely of surface normals. Each point in the image memory has 12 bits of information about the surface normal, which can be used to look up in the color map the brightness of that point on the surface under current assumptions of surface characteristics and lighting.

No matter how complicated the calculation (perhaps there are several light sources of different colors and the objects have certain transparency and specularity properties), it need be done only once in this situation for the 2**12 = 4096

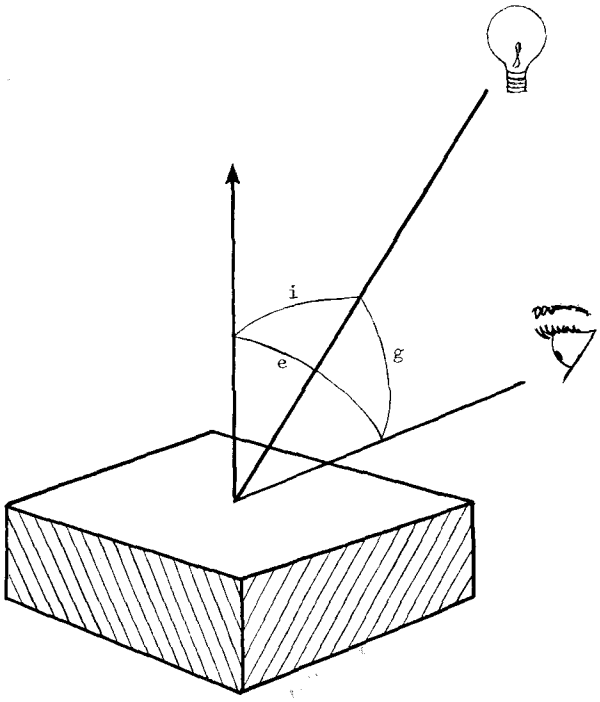

FIG. 6. Important angles in calculating the perceived brightness of a surface. The arrow is the normal to the surface; i is the angle of incidence, e the angle of emittance, and g the phase angle.

separate surface characteristics. Since the color map transformation is independent of pixel location, shadows cannot be calculated, and the light sources must be at infinity. Instantiated in the color map, the results of the calculation are applied automatically and continuously to the intrinsic image, resulting in a viewable shaded image.

The use of storage for intrinsic properties is completely unconstrained—for example, it may be encoded to use the limited pixel size intelligently; but a straightforward example will be used to illustrate the ideas. Assume that we want to display a static scene of solid, opaque objects, under varying lighting and surface reflectance conditions, and that the shading calculation depends on these characteristics and the surface normal of the objects.

We store in image memory the orientation of the surface seen at each pixel in the image. The orientation can be represented by a unit surface-normal vector (x, y, z). Since the objects are opaque, the range of this vector covers the half-sphere with z (the viewing direction) ≥ 0. Since it is a unit vector, we have

$$z = \text{sqrt}(1 - x**2 - y**2).$$

Hence, we need only store its x and y components. Allowing 6 bits each, let the components of these vectors be quantized to lie in $[-31, 31]$. The color map address corresponding to the surface normal $[x, y, z]$ may then be $(x + 31)*(2**6) + (y + 31)$. For a Lambertian surface, the brightness of a point is proportional

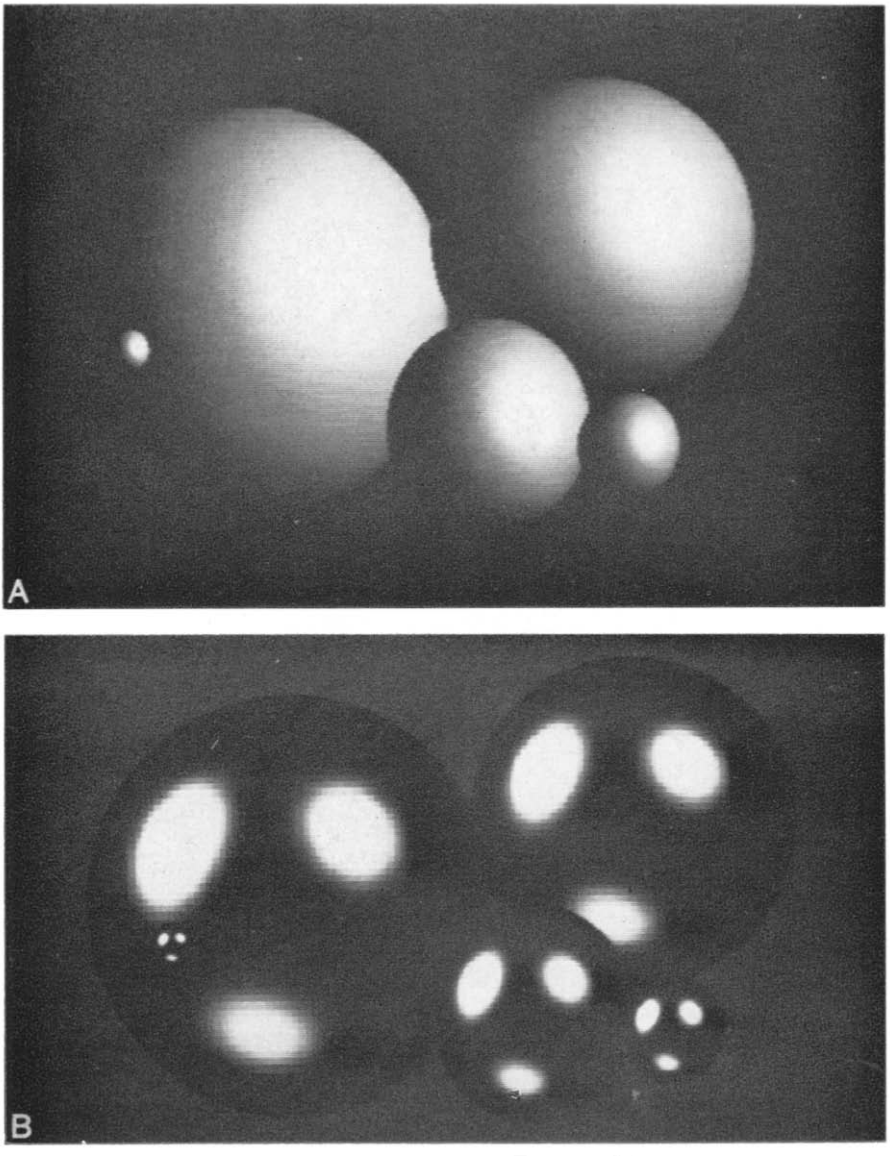

FIG. 7. A scene of spheres shown with various lighting and specularity conditions. The color map was rewritten for these effects; the image memory was not. The program producing these effects extends to multiple colored light sources of differing intensities.

to the cosine of the angle between the surface normal and the incident light (Fig. 6) (Horn and Bachman [6]). With a light source at infinity in the direction $[A, B, C]$, the color map entry for address $(x + 31)*(2**6) + (y + 31)$ is therefore proportional to the dot product:

$$A*x + B*y + C*(\text{sqrt}((31**2) - x**2 - y**2))).$$

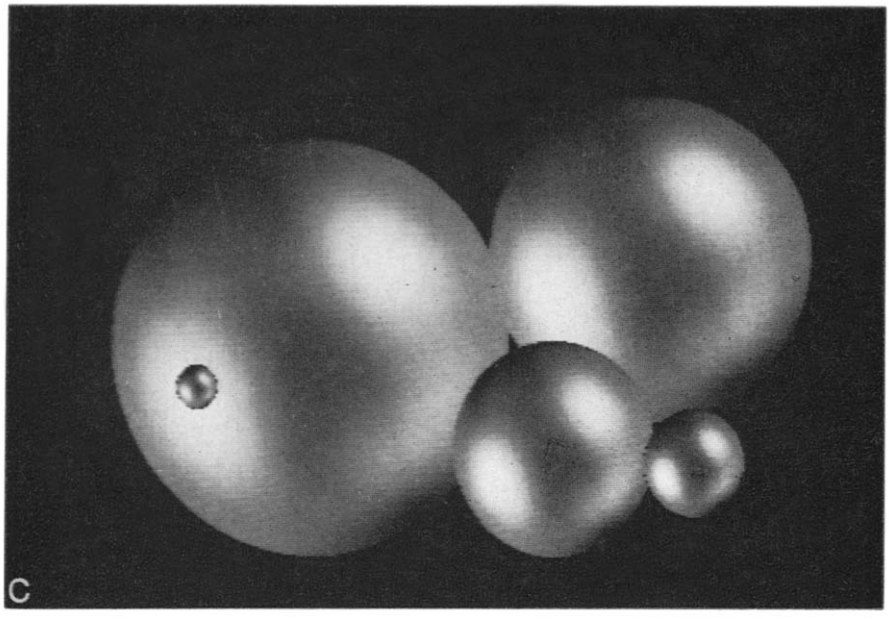

FIG. 7—*Continued.*

Other surface characteristics (specularity, color) and lighting conditions (many lights, colored lights) are straightforward (see Fig. 7), and different encodings of the unit vectors are possible (the above scheme only uses $pi/4$ of the available 4096 pixel values) (Bass [16]).

3.6. Dynamic Display Options for Polyhedra

The display of polyhedra, as wire frames and as solid objects, is a classic problem in computer graphics (Sutherland *et al.* [14]). There are many ways in which these figures can be displayed, each with its own advantages. If the edges are the important features, then a wire frame display is appropriate. When the surfaces are important, these should be shown, and the edges suppressed. In either case, there remains the question of hidden lines (surfaces). Usually, these should be removed. Sometimes, especially with wire frame displays, the "hidden" lines must be shown, typically with a distinguished line style. Thus, we have four distinct display modes: lines or surfaces, with or without hidden components.

If our display needs are fixed, then we can make the two decisions outlined above and paint an appropriate picture on the display screen. Suppose, however, that we need to switch quickly among the various display modes. The color map can be used to perform this switching, without changing the image memory.

In order to do this, the image memory is used to hold brightness information for the pixels (provided by some program) and, for each pixel, to code bits indicating whether the pixel is internal to a face, or if on an edge, whether the edge is hidden or not. Once the image memory is written with this coded information, it is a

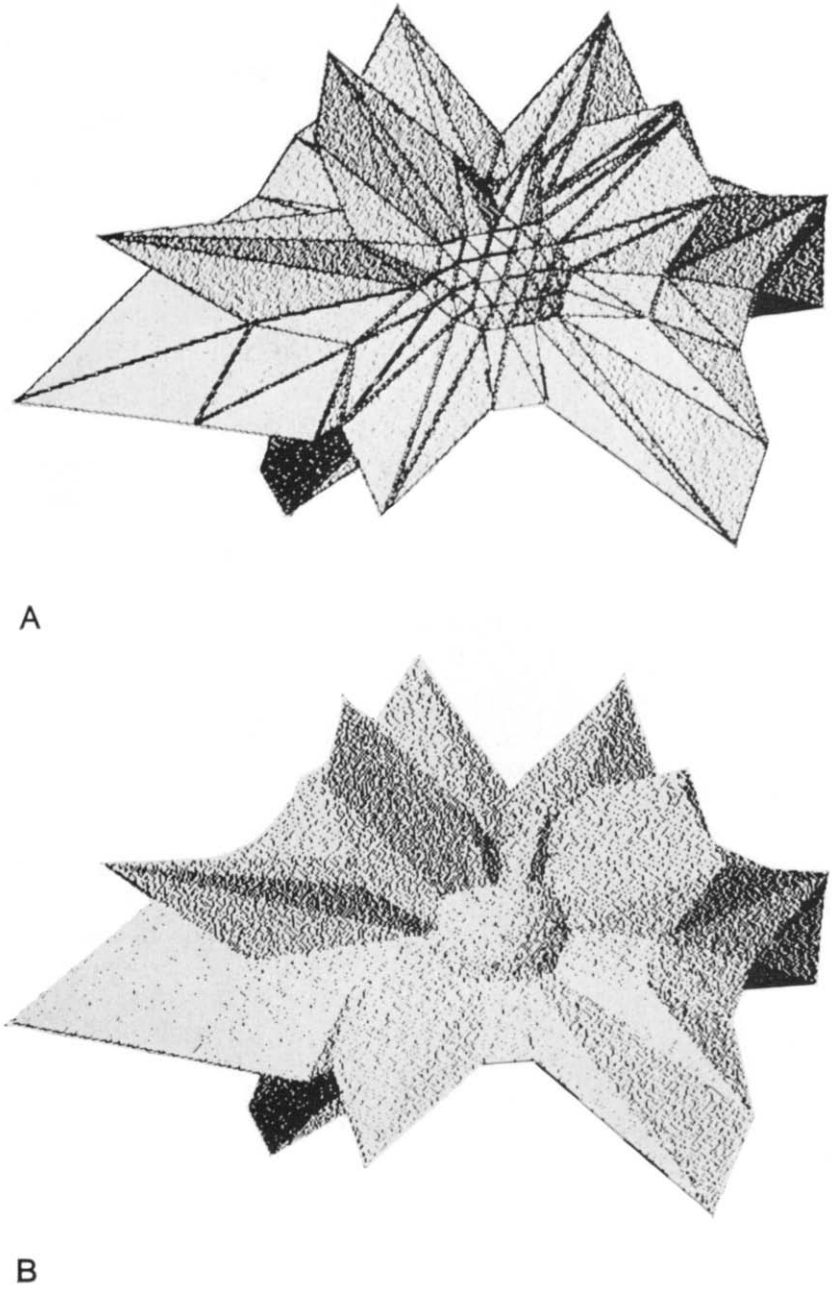

FIG. 8. A shaded polyhedron shown with (A) and without (B) lines at edges of faces. The shading was done with a halftone technique.

simple matter to display only that part of the picture which is currently of interest. We can even display only the hidden lines and suppress the visible edges.

Once again, the basic problem lies in the allocation of image-memory bits.

For example, let us assume that we have a simple solid polyhedron. We can use one bit to mean "edge," one bit to mean "hidden," and the remaining $(i - 2)$ bits for shading the visible surface. This will allow us to select the visible surface, the surface with visible edges intensified, the surface with all edges, or the edges alone.

Note that we cannot retain information about the shading of points on hidden surfaces, and (depending on our conventions) either visible or hidden edges will be masked when two edges intersect on the display screen. Some of these problems can be solved by using more "code" bits and fewer shading bits for simple objects, but this is probably impractical.

Figure 8 indicates a difference in information provided by making edges explicit. The objects are of the well-tessellated class described by Brown [5] which has good properties for raster-graphics display.

3.7. Color System Transformations

The RGB monitor included in our sample hardware configuration displays color information in a format which is appropriate for presentation to the human retina. It is well established that this (R, G, B) three-space is an adequate model for human low-level color sensation. However, often this coordinate system is not the most useful one for applications other than direct display.

Depending upon various image-processing needs, color images may be represented in a completely different color space. For example, if the image is to be printed, as well as displayed on a CRT, then it may be represented in a "negative" color space, such as cyan–magenta–yellow. If it is to be used in a scene analysis system, then the possibilities increase. Recent scene analysis research involving color images has involved the use of many different color spaces (Joblove [8]; Kender [9]; Sloan and Bajcsy [13]; Smith [15]).

A few of the characteristics of some common color systems are sketched below. The RGB space may be displayed in the well-known color cube. When a color made by mixing three other colors (called primaries) is normalized by dividing the primary components by the brightness of the color, the resulting colors lie in a plane in a chromaticity diagram. The chromaticity diagram, which includes all normalized colors visible to humans, is best approximated with three primaries by choosing them to be particular hues of red, green, and blue. This explains why most color monitors are based on the RGB space, and why RGB is therefore a natural choice for low-level communication with (unmapped) raster-graphics devices. In addition, shading for a color surface illuminated by a single light is obtained by interpolating between the colors and black.

YIQ space is used in color television transmission and is a linear transformation of RGB space. The transform from RGB to YIQ is:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.30 & 0.59 & 0.11 \\ 0.60 & -0.28 & -0.32 \\ 0.21 & -0.52 & 0.31 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

The Y, or luminance component, is what a black-and-white TV responds to in a

color signal; it was chosen on psychophysical grounds to be an acceptable monochromatic version of the color image.

HSV (hue, saturation, value) space is based on a polar coordinate rather than a Cartesian system. Roughly, saturation and value measure how much a given hue departs from white or black, respectively, while hue indicates "pure color" (e.g., yellow or blue-green). HSV space is useful for artists who think of obtaining tints or shades (generally tones) of color by mixing white or black with a pure color. Further, it happens that interpolating between a color and an unsaturated blue in HSV simulates the effect of atmospheric scattering more closely than would similar interpolation in (say) RGB. Interpolation in HSV space can easily preserve saturation between hues; this is often what is required, but is difficult in RGB space.

When color naming is important, a congenial space is that of opponent processes. This affine transformation of RGB has three axes: white minus black, yellow minus blue, and red minus green. This space is

$$\begin{bmatrix} W - BLA \\ Y - BLU \\ R - G \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & \frac{2}{3} & \frac{2}{3} \\ \frac{1}{2} & \frac{1}{2} & -1 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

This space captures the fact that we can say, e.g., "dark reddish-blue" but not "lightish dark" or "yellowish blue."

Many more color spaces have found use in practice, and it seems likely that a varied arsenal of them will be useful for sophisticated color graphics use. In order to display images in non-RGB systems on an RGB monitor, we must transform them to the RGB color space. The color map is ideal for this application. The transformation from the color space of one's choice into the RGB space need be computed only once, and kept in storage local to the display processor. A library of such transformations can be created and used repeatedly. The transformations may be arbitrarily complex, as long as they are point-by-point. That is, the transformed value must depend only upon the value of a single pixel, and may not be influenced by a larger neighborhood. This condition is met by most of the color systems used in practice. Even when this condition is not met, there may be an approximate transformation meeting the condition which is sufficient for display purposes.

For example, assume that we have a color image represented in the YIQ color space. In practice, the I and Q values will probably be shifted into the positive range and spread out by a small linear scale (Nagin *et al.* [10]). Without the color map, we would be forced to "false-color" the image, say by assigning Y values to the red gun, I to green, and Q to blue. With the color map, on the other hand, we can reverse the linear transformation above, and turn our RGB monitor into a YIQ monitor. If we have several YIQ images, we need to compute this transformation only once, saving the resulting look-up table in local memory.

4. CONCLUSION

Full color raster-graphics displays are capable of presenting complex images, which require large amounts of storage, bandwidth, and computation. Many

interesting display applications involve transformations which can be viewed as reinterpretations of the image-memory data. In the (usual) case in which the number of pixels in the image is much greater than the number of possible values each pixel may have, a color map is an effective way to implement these transformations.

We have presented several representative applications, along with techniques for using the color map in these domains.

ACKNOWLEDGMENTS

This work was supported in part by Alfred P. Sloan Foundation Grant 74-12-5 and in part by National Science Foundation Grant MCS76-10825. Eugene Ball wrote the DEATHSTAR game and color overlay utilities, Dan Bass wrote the utilities for use with surface-normal and colored light intrinsic images, and a colored shaded spheres and blocks test program. Peter Selfridge assisted with the well-tessellated surface code. The referees made many helpful suggestions.

REFERENCES

1. W. I. Badler and L. C. Miller, *A Formal Model for Pseudo-color Selection,* Technical report, Computer and Information Sciences Department, University of Pennsylvania, June 1977.
2. H. G. Barrow and J. M. Tenenbaum, *Recovering Intrinsic Scene Characteristics from Images.* Technical Note 157, SRI International, April 1978.
3. J. F. Blinn, Raster graphics, Tutorial notes, SIGGRAPH, 1978.
4. J. M. Booth and R. B. Schroeder, Design considerations for digital image processing systems, *Computer* **10**, No. 8, 1977.
5. C. M. Brown, *Fast Display of Well-Tessellated Surfaces,* TR23, University of Rochester Computer Science Department, March 1978.
6. B. K. P. Horn and B. L. Bachman, Using synthetic images to register real images with surface models, *Comm. ACM* **21**, 1978, 914–924.
7. R. A. Hummel, Histogram modification techniques, *Computer Graphics and Image Processing* **4**, 1975, 209–224.
8. G. H. Joblove and D. Greenberg, Color spaces for computer graphics, *Computer Graphics* **12**, 1978, 20–25.
9. J. Kender, *Saturation, Hue, and Normalized Color Calculation, Digitization Effects and Use,* Carnegie–Mellon University, 1976.
10. P. A. Nagin, A. R. Hanson, and E. M. Riseman, *Region Extraction and Description Through Planning,* COINS Technical Report 77-8, University of Massachusetts at Amherst, 1977.
11. N. Negroponte, Raster scan approaches to computer graphics, *Computers and Graphics* **2**, 1977.
12. D. Nitzan, A. E. Brain, and R. C. Duda, The measurement and use of registered reflectance and range data in scene analysis, *Proc. IEEE* **65**, 1977, 206–220.
13. K. R. Sloan and R. Bajcsy, A computational structure for color perception, in *Proceedings, ACM 75, Minneapolis, 1975,* 42–45.
14. I. E. Sutherland, R. F. Sproull, and R. A. Schumaker, A characterization of ten hidden-surface algorithms, *Comput. Surveys* **6**, 1974, 1.
15. A. R. Smith, Color gamut transform pairs, *Computer Graphics* **12**, 1978, 12.
16. D. H. Bass, Using the color map for reflectivity calculations: Specific techniques and graphic results, *Computer Graphics and Image Processing,* submitted.