Robert Ulichney

Video Rendering

Video rendering, the process of generating device-dependent pixel data from device-independent sampled image data, is key to image quality. System components include scaling, color adjustment, quantization, and color space conversion. This paper emphasizes methods that yield high image quality, are fast, and yet are simple and inexpensive to implement. Particular attention is placed on the derivation and analysis of new multilevel dithering schemes. While permitting smaller frame buffers, dithering also provides faster transport of the processed image to the display—a key benefit for the massive pixel rates associated with full-motion video.

Perhaps the most influential characteristic governing the perceived value of a system that displays images is the way the pictures look. Image appearance is largely dependent upon the quality of rendering, that is, the process of taking device-independent data and generating device-dependent data tailored for a particular target display.

The topic of this paper is the processing of sampled image data and not synthetic graphics. For graphics rendering, primitives such as specifications of triangles are converted to displayable picture elements or pixels. The atomic elements handled by a video rendering system are device-independent pixels. Whereas a prerendered graphics image can be compactly represented as a collection of triangle vertices, prerendered video achieves compaction by means of compression techniques.

Sampling broadcast video requires a data rate of more than 9 million color pixels per second; the need of some relief for storage and networks is clear. Video compression reduces redundancy in the source image and thereby reduces the amount of data to be transmitted. Dramatic reductions in data rate can be achieved with little degradation in image quality. The Joint Photographic Experts Group (JPEG) standard for still frame and the Motion Picture Experts Group (MPEG) and Px64 standards for motion video are current committee compression techniques.[1] Several other nonstandard schemes exist, including a simple compression method conducive to software-only implementation.[2]

Video rendering receives decompressed image data as input. Since every decompressed pixel must be processed, speed is essential. This paper focuses on rendering methods that are fast, simple, and inexpensive to implement. Performance at video rates can be achieved with minimal hardware or even software-only solutions.

The Rendering Architecture section reviews the components of a rendering system and examines design trade-offs. The paper then presents details of new and efficient dithering implementations. Finally, video color mapping is discussed.

Rendering Architecture

Figure 1 illustrates the major phases of a video rendering system: (1) filter and scale, (2) color adjust, (3) quantize, and (4) color space convert.

In the first stage, the original image data must be resampled to match the target window size. A separate scaling system should be used for the horizontal and vertical directions to handle the case where the pixel aspect ratio must be changed. For example, such asymmetric scaling is needed when the target display expects square pixels and the original pixels are not square.

The best filters to use in combination with scaling have been determined from a perceptual point of view.[3] When limiting the bandwidth to reduce the data rate, a Gaussian filter with a standard deviation $\sigma = 0.30 \times$ output period is recommended. For interpolation, the filter preferred (because the filtered results looked most like the original) was a cascade of two: first, sharpen with a Laplacian filter, and second, follow by convolution with a Gaussian filter with $\sigma = 0.375 \times$ input period.

A typical sharpening scheme can be expressed by the following equation:

$$I_{\text{sharp}}[x,y] = I[x,y] - \beta\Psi[x,y] * I[x,y], \quad (1)$$

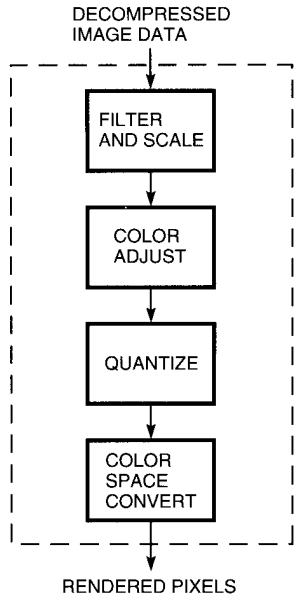


Figure 1 Image Rendering System

where $I[x,y]$ is the input image, $\Psi[x,y]$ is a digital Laplacian filter, and "$*$" is the convolution operator.[4] The nonnegative parameter β controls the degree of sharpness, with $\beta = 0$ indicating no change in sharpness. When enlarging, sharpening should occur before scaling, and when reducing, sharpening should take place after scaling. The filtering discussed here is assumed to be two-dimensional, which requires image line buffering. For economy, horizontal-only filtering is sometimes used.

The simplest means of scaling is known as nearest-neighbor scaling, and its simplest implementation is based on the Bresenham scan conversion algorithm for drawing straight lines.[5] This algorithm can be applied to image scaling and performed with only three registers and one adder.[6] Further optimizations make this algorithm especially suited for real-time use.[7]

The second stage of rendering is color adjust, most easily achieved with a look-up table (LUT). Each color component uses a separate adjust LUT. In the case of a luminance-chrominance color, an adjust LUT for the luminance component controls contrast and brightness, and LUTs for the chrominance components control saturation.

For so-called true-color frame buffers with 24-bit depths, visual artifacts that can result from insufficient amplitude resolution do not occur. With smaller frame buffers, restricting the amplitude of the color components red, green, and blue (RGB) with a simple uniform quantizer causes false contours to appear in slowly varying regions. This issue leads to the third stage in the rendering system, quantization.

The three basic classes of techniques for circumventing the problem of insufficient colors or color memory are (1) histogram-based methods, (2) chrominance-subsampled frame buffers, and (3) dithering. All histogram-based methods, sometimes called palette selection, require two passes of the entire image data: the first to acquire the histogram statistics to fabricate a three-dimensional quantizer to N colors and the second to perform the pixel assignments. Perhaps the fastest method is the popularity algorithm, where a simple sort finds the N colors with the highest frequency, and all other colors are mapped to those.[8]

A more compute-intensive method, but one that in general performs much better, is the often-used, median-cut algorithm.[8] In this method, the color space is repeatedly subdivided into smaller rectangular solids at the median planes, with the goal that each of the selected colors represent an equal number of colors in the image. The average of the colors in each of the final regions is the color used in the quantizer. A later, less compute-intensive variation is the mean-split algorithm. Also, several clustering techniques have been reported that result in less quantization error than the above-mentioned methods. One method, for example, minimizes the sum of the squares of the errors.[9] In all cases, however, color problems can occur in other application windows because each frame requires a different color map; the colors in the other windows become scrambled in a different way for each color map.

One advantage of representing image data in a luminance-chrominance space is that chrominance requires less spatial resolution than luminance to achieve excellent image quality. Visual perception of differences in chrominance is much less than that for luminance. The television standards have been exploiting this fact for decades. The quantization approach of using chrominance-subsampled frame buffers is built on this fact, deferring conversion to the RGB components until just after the data is read for display.[10,11,12]

Typical implementations of chrominance-subsampled frame buffers average each of the two chrominance values in a given luminance-chrominance color representation over a region that is either 2 by 2 or 4 by 4 pixels. Assuming 8 bits

of amplitude resolution per color component, the 2-by-2-pixel case results in an average of ((2 × 2 × 8 luminance bits) + (8 + 8 chrominance bits))/(2 × 2 pixels) or 12 bits per pixel; similarly, the 4-by-4-pixel case results in 9 bits per pixel. This approach requires expensive hardware to up-sample the chrominance components and convert the color space at video rates. These nonstandard frame buffers can also cause severe incompatibility problems with most applications that expect RGB frame buffers. While chrominance subsampled frame buffers can accommodate most sampled natural images, thin-line graphics can be annihilated.

The third alternative for quantization is to use a dithering method. Several methods exist that are designed primarily for binary output, but all are extendable to multilevel color.[4,13,14,15] A "level" is a shade of gray, from black to white, or a shade of a color component, from black to the brightest value. The basic principle of dithering is to use the available subset of colors to produce, by judicious arrangement, the illusion of any color in between.

Although neighborhood operations, most notably error diffusion, produce good-quality dithering, they are computationally complex and require additional storage. For video processing, where speed is essential, we turned our focus to those dithering methods that are point operations, that is, methods that operate on the current pixel only without considering its neighbors. Each color component of every pixel in the image has an associated "noise" or dither amplitude that is added to it before that component is passed to a uniform quantizer.

Historically, the first dithering method used for video processing was white noise dithering, where a pseudorandom number was added to each luminance value before quantization. This method was practiced soon after the dawn of television.[16] However, the low-frequency energy in white noise causes undesirable textures and graininess.

A preferred method is the point process of ordered dithering, where a deterministic noise array tiles the plane in a periodic manner. Dither arrays can be designed to minimize low-frequency texture. The most popular are the so-called recursive tessellation arrays.[17,18] These arrays yield results superior to those of white noise dithering but suffer from structured rectangular patterns.

A new ordered dither array design, called the "void-and-cluster" method, eliminates both the low-frequency textures of white noise and the rectangular patterns of recursive tessellation arrays.[19] The name describes the dither array design process in which voids and clusters are located and mitigated.

For the high-speed case of motion video, an ordered dithering scheme has important advantages over chrominance-subsampled frame buffers and histogram-based approaches. Quantization by dithering allows the use of conventional frame buffers, does not require the time-consuming process of making two passes over each frame (or every Nth frame), does not cause other applications to change color maps with every Nth frame, and allows any number of colors to be selected at render time. Also, experiments have shown that the image quality achieved by dithering is very competitive with the other methods, when compared over a range of sample images. Even when 24-bit frame buffers are available, the increased speed of loading three or four 8-bit color pointers or index values in the time required to load a single 24-bit pixel makes dithering a viable alternative in the design of desktop video systems.

By way of comparison, Figure 2 illustrates some of the methods described in this section. A 240-by-360-pixel, 8-bit monochrome image was rendered to only two levels and displayed at 100 dots per inch (dpi). Figure 2a depicts an image that was dithered with white noise; in Figure 2b, the same image was dithered using an 8-by-8 recursive tessellation dither array; and Figure 2c shows the image dithered with the new 32-by-32 void-and-cluster array. To illustrate the effect of sharpening, Figure 2d shows the image in Figure 2c presharpened using a digital Laplacian filter as in equation (1), with a sharpening factor of $\beta = 2.0$. The goal of this coarse example is to amplify the different effects. The same methods apply to multilevel and color output, where the resulting quality is much higher.

Fast Multilevel Dithering

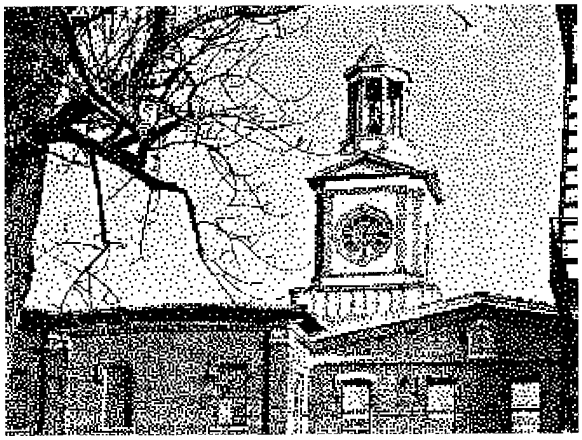
This section presents the details of simple, yet powerful new designs to perform multilevel ordered dithering. The simplicity of these methods allows for implementation with minimum hardware or software only, yet guarantees output that preserves the mean of the input. The designs are flexible in that they allow dithering from any number of input levels N_i, to any number of output levels N_o, provided $N_i \geq N_o$. Note that N_i and N_o are not restricted to be powers of two.

Each color component of a color image is treated as an independent image. The input image I_i can have values

(a) *Dither with a White Noise Threshold*

(b) *Dither with an 8-by-8 Recursive Tessellation Threshold Array*

(c) *Dither with a 32-by-32 Void-and-cluster Threshold Array*



(d) *Same as (c) with Laplacian Sharpening,* $\beta = 2.0$

Figure 2 Examples of Rendering to Two Output Levels

$$L_i \in \{0,1,2,\ldots,(N_i - 1)\},$$

and the output image L_o can have values

$$L_o \in \{0,1,2,\ldots,(N_o - 1)\}.$$

A deterministic dither array of size $M \times N$ is used that is periodic and tiles the entire input image. To simplify addressing of this array, M and N should each be powers of two. A dither template defines the order in which dither values are arranged. The elements of the dither template T have values

$$T \in \{0,1,2,\ldots,(N_t - 1)\},$$

where N_t is the number of template levels, which represent the levels against which image input values are compared to determine their mapping to the output values. The dither template is thus central to determining the nature of the resulting dither patterns.

Figure 3 shows a dithering system that comprises two memories and an adder. The system takes an input level L_i at image location $[x,y]$ and produces output level L_o at the corresponding location in the dithered output image. The dither array is addressed by x' and y', which represent the low-order bits of

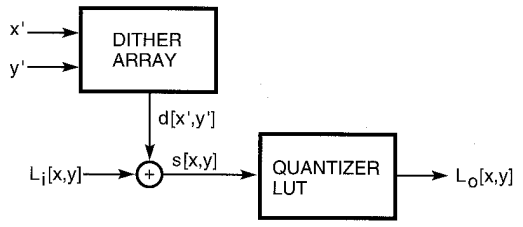


Figure 3 Dithering System with Two LUTs

the image address. The selected dither value $d[x',y']$ is added to the input level to produce the sum s. This sum is then quantized by addressing a quantizer LUT to produce the output level L_o.

The trick to achieving mean-preserving dithering is to properly generate the LUT values. The dither array is a normalized version of the dither template specified as follows:

$$d[x',y'] = \text{int}\{\Delta_d(T[x',y'] + \tfrac{1}{2})\}, \quad (2)$$

where Δ_d, the step size between normalized dither values, is defined as

$$\Delta_d = \frac{\Delta_Q}{N_t} \quad (3)$$

and Δ_Q is the quantizer step size

$$\Delta_Q = \frac{(N_i - 1)}{(N_o - 1)}. \quad (4)$$

Note that Δ_Q also defines the range of dither values. The quantizer LUT is a uniform quantizer with N_o equal steps of size Δ_Q.

The precise expressions in equations (2), (3), and (4) were arrived at through extensive analysis of the average output resulting from processing input images of a constant value, over a wide range of N_i, N_o, and N_t.

One-memory Dithering System

Using the above expressions, it is possible to simplify the system by exchanging one degree of freedom for another. A bit shifter can replace the quantizer LUT at the expense of forcing the number of input levels N_i, to be set by the system. For hardware implementations, this design affords a considerable cost reduction.

The system and method of Figure 3 assume that N_i is given as a fixed parameter, as is usually the case with most imaging systems and file formats. However, for image sources such as hardware that generates synthetic graphics, arbitrarily setting N_i often has no effect on the amount of computation involved. If an adjust LUT is used to modify the image data, including a gain makes a "modified adjust LUT." Figure 4 depicts such a system, where L_r is the raw input level. The unadjusted or raw input image can have the values

$$L_r \in \{0,1,2,\ldots,(N_r - 1)\},$$

where N_r is the number of raw input levels, typically 256. Therefore, the modified adjust LUT must impart a gain of

$$\frac{N_i - 1}{N_r - 1}.$$

To solve for N_i, recall that in the method of Figure 3 the quantizer was defined to have equal steps of size Δ_Q as defined in equation (4). The quantizer LUT can be replaced by a simple R-bit shifter, if the variable Δ_Q can be forced to be an exact binary number,

$$\Delta_Q = 2^R. \quad (5)$$

N_i can then be set by the expression

$$N_i = (N_o - 1)2^R + 1. \quad (6)$$

The integer R is the number of bits the R-bit shifter shifts to the right to achieve quantization. Specifying R in terms of N_o, equation (6) becomes

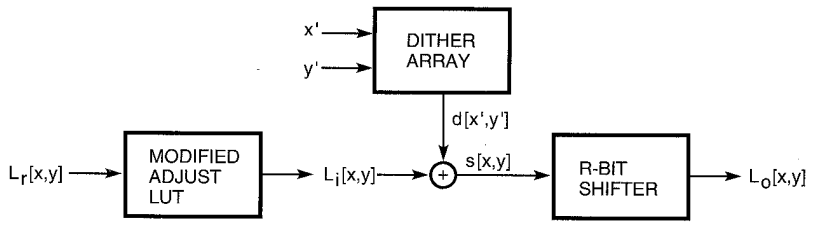


Figure 4 One-memory Dithering System with an Adjust LUT and Bit Shifter

$$R = \log_2 \frac{(N_i - 1)}{(N_o - 1)}. \quad (7)$$

To completely specify this problem requires specifying the range for N_i. It is reasonable to do this by specifying the number of bits b by which the image input values are to be represented. Specifying b limits N_i to the range

$$2^{b-1} < N_i \leq 2^b. \quad (8)$$

Parameter b will be a key value in specifying the resulting system.

Given the two expressions, (7) and (8), and the two unknowns, R and N_i, a unique solution exists because the range of N_i is less than a factor of two, and R and N_i are integers. To solve for R, substitute equation (6) for N_i in equation (8). The resulting equation is

$$2^{b-1} < (N_o - 1)2^R + 1 \leq 2^b \quad (9)$$

or

$$\log_2\left(\frac{2^{b-1} - 1}{N_o - 1}\right) < R \leq \log_2\left(\frac{2^b - 1}{N_o - 1}\right). \quad (10)$$

Since $2 \leq N_o \leq N_i$, the range of the expression in equation (10) must be less than one. Hence, given that R is an integer,

$$R = \text{int}\left\{\log_2\left(\frac{2^b - 1}{N_o - 1}\right)\right\}. \quad (11)$$

N_i in equation (6) is now specified.

As an example, consider the case where N_o equals 87 (levels), b equals 9 (bits), N_t equals 1,024 (levels, for a 32-by-32 template), and N_r equals 256 (levels). Thus, R equals 2, and the R-bit shifter drops the least-significant 2 bits. N_i equals 345 (levels); the dither array is normalized by equation (2) with $\Delta_d = 1/256$; and the gain factor to be included in the modified adjust LUT is 344/255. This data is loaded into the system represented by Figure 4 and uniformly maps input pixels across the 87 true output levels, giving the illusion of 256 levels.

The output image that results from either of the dithering systems illustrated in Figure 3 or Figure 4 appears to contain more effective levels than are actually displayed. An effective level is either a perceived average level that is dithered between two true output levels or shades or an actual true output level. A small number of template levels N_t dictates the resulting number of effective levels. When N_t is large, the number of effective levels is equal to the number of input levels N_i, because it is not possible to display more effective outputs than inputs. More precisely,

$$\text{Effective Levels} = \begin{cases} (N_o - 1)N_t + 1 & \frac{\Delta_Q}{N_t} > 1 \\ N_i & \frac{\Delta_Q}{N_t} \leq 1. \end{cases} \quad (12)$$

Note that Δ_Q/N_t in equation (12) is equal to Δ_d. When $\Delta_d < 1$, the normalization of the dither array, i.e., equation (2), results in integer truncated values that are not all unique. At this point, the number of effective levels saturates to N_i.

Data Width Analysis

The design of an efficient dithering system, particularly in hardware, depends on knowing the number of bits required for all data paths in the system. This section presents an analysis of the one-memory dithering system shown in Figure 4.

The system input b, i.e., the bit depth of the image input values, limits the data path for $L_i[x,y]$. The analysis shows the derivation of the precise bit depths for the other data paths. In summary, the derivation proves that the dither values in the dither matrix memory require R bits, where $R_{max} = (b - 1)$ and $s = L_i + d$ (and thus the R-bit shifter) require only b bits.

Bits Needed for Dither Matrix The amount of memory needed to store the dither matrix is an important concern; d_{max} denotes the maximum value. To determine d_{max}, substitute the maximum value of $T[x',y']$, which is $(N_t - 1)$, into equation (2). The resulting equation is

$$d_{max} = \text{int}\left\{\frac{2^R}{N_t}((N_t - 1) + \frac{1}{2})\right\}$$
$$= \text{int}\left\{2^R\left(\frac{N_t - \frac{1}{2}}{N_t}\right)\right\}. \quad (13)$$

d_{max}, which depends on N_t, thus has a value in the range

$$2^{R-1} \leq d_{max} \leq 2^R - 1. \quad (14)$$

For the case of a dither matrix with one value, namely $N_t = 1$, d_{max} equals the lower end of this range. d_{max} equals the high end of the range for large dither matrices, where $2^{R-1} \leq N_t$. An important observation is that for all values in the range of expression (14), the number of bits needed is exactly R.

From equation (11), the value of R increases as N_o decreases. The smallest possible value of N_o is 2, which is for bitonal output. In this case, the maximum value of R is

$$R_{max} = \text{int}\{\log_2(2^b - 1)\} = (b - 1). \quad (15)$$

So, the number of bits needed for the dither values is R, which can be as large as $(b - 1)$.

Bit Width of Adder Recall that $s[x,y] = L_i[x,y] + d[x,y]$. The number of bits needed for this sum determines the size of the adder and the size of the R-bit shifter. L_i can be at most $(N_i - 1)$ and, as determined in the last section, d_{max} can be at most $(2^R - 1)$. So,

$$s_{max} = (N_i - 1) + (2^R - 1). \quad (16)$$

From equation (6),

$$(N_i - 1) = (N_o - 1)2^R, \quad (17)$$

which gives

$$s_{max} = 2^R(N_o - 1) + (2^R - 1) = 2^R N_o - 1. \quad (18)$$

We can express R in terms of N_o by using equation (11):

$$R = \text{int}\{\log_2(2^b - 1) - \log_2(N_o - 1)\}. \quad (19)$$

Each of the two terms in the equation (19) can be expressed in terms of an integer part and a fractional part:

$$\log_2(2^b - 1) = (b - 1) + \epsilon_1, \quad (20)$$

where

$$0 < \epsilon_1 < 1,$$

and

$$\log_2(N_o - 1) = K + \epsilon_2, \quad (21)$$

where K is an integer, and

$$0 \le \epsilon_2 < 1.$$

Now equation (19) can be rewritten as

$$R = (b - 1) - K + \text{int}\{\epsilon_1 - \epsilon_2\}. \quad (22)$$

ϵ_2 is largest when N_o (an integer) is a large power of 2. Because N_o cannot be greater than N_i,

$$2^b \ge N_o.$$

This fact, combined with equations (20) and (21), yields the further condition

$$\epsilon_1 \ge \epsilon_2.$$

Therefore, $\text{int}\{\epsilon_1 - \epsilon_2\}$ in equation (22) must be equal to zero, and the value of R becomes

$$R = b - 1 - K. \quad (23)$$

We can express N_o in equation (18) in terms of the same integer K of equation (21) by noting that

$$\log_2 N_o = K + \epsilon_3, \quad (24)$$

where

$$0 < \epsilon_3 \le 1. \quad (25)$$

Observe that ϵ_3 is equal to 1, where N_o is an exact power of 2. Substituting

$$N_o = 2^{K+\epsilon_3}$$

and equation (23) into equation (18) gives

$$s_{max} = 2^{b-1-K}\,2^{K+\epsilon_3} - 1 = 2^{b-1+\epsilon_3} - 1. \quad (26)$$

Because of the range of ϵ_3 in equation (25), the range of s_{max} must be

$$2^{b-1} - 1 < s_{max} \le 2^b - 1, \quad (27)$$

which requires exactly b bits.

As a check, the size of the shift register should equal the number of bits required for N_o plus R. The number of bits needed for N_o is

$$\text{int}\{1 + \log_2(N_o - 1)\}. \quad (28)$$

Using the expression in equation (21), this value becomes

$$\text{int}\{1 + K + \epsilon_2\} = K + 1. \quad (29)$$

So, the size of the shift register must be

$$(K + 1) + (b - 1 - K) = b \text{ bits},$$

which matches the maximum size of the sum s.

Color Space Conversion

Referring once again to Figure 1, consider the final subsystem of a video rendering system—color space convert. Assuming a frame buffer that is expecting RGB data, color space conversion is not necessary if the source data is already represented in RGB, as in the case of graphics generation systems. However, motion video is essentially always transmitted and stored in a luminance-chrominance space. Such a representation allows subsampling of the chrominance, as mentioned earlier, which reduces bandwidth requirements; all video standards exploit this method of bandwidth reduction. It is also more intuitive to color adjust in a luminance-chrominance space.

Prior to proceeding to the quantize subsystem shown in Figure 1, all color components must be at the same final spatial resolution for a dithering method to work correctly. Chrominance components, then, need to be up-sampled to the same rate as luminance components.

Although the chromaticities of the RGB primaries of the major television standards vary slightly, all television systems transmit and store the color data in YUV space. Y represents the achromatic component that is loosely called the luminance component. (The term luminance has a specific photometric definition that is not what is represented in a video Y component.) U and V are color difference components, where U is proportional to (Blue − Y) and V is proportional to (Red − Y).

Figure 5 is an orthographic projection of YUV space. Inside the YUV rectangular solid is the

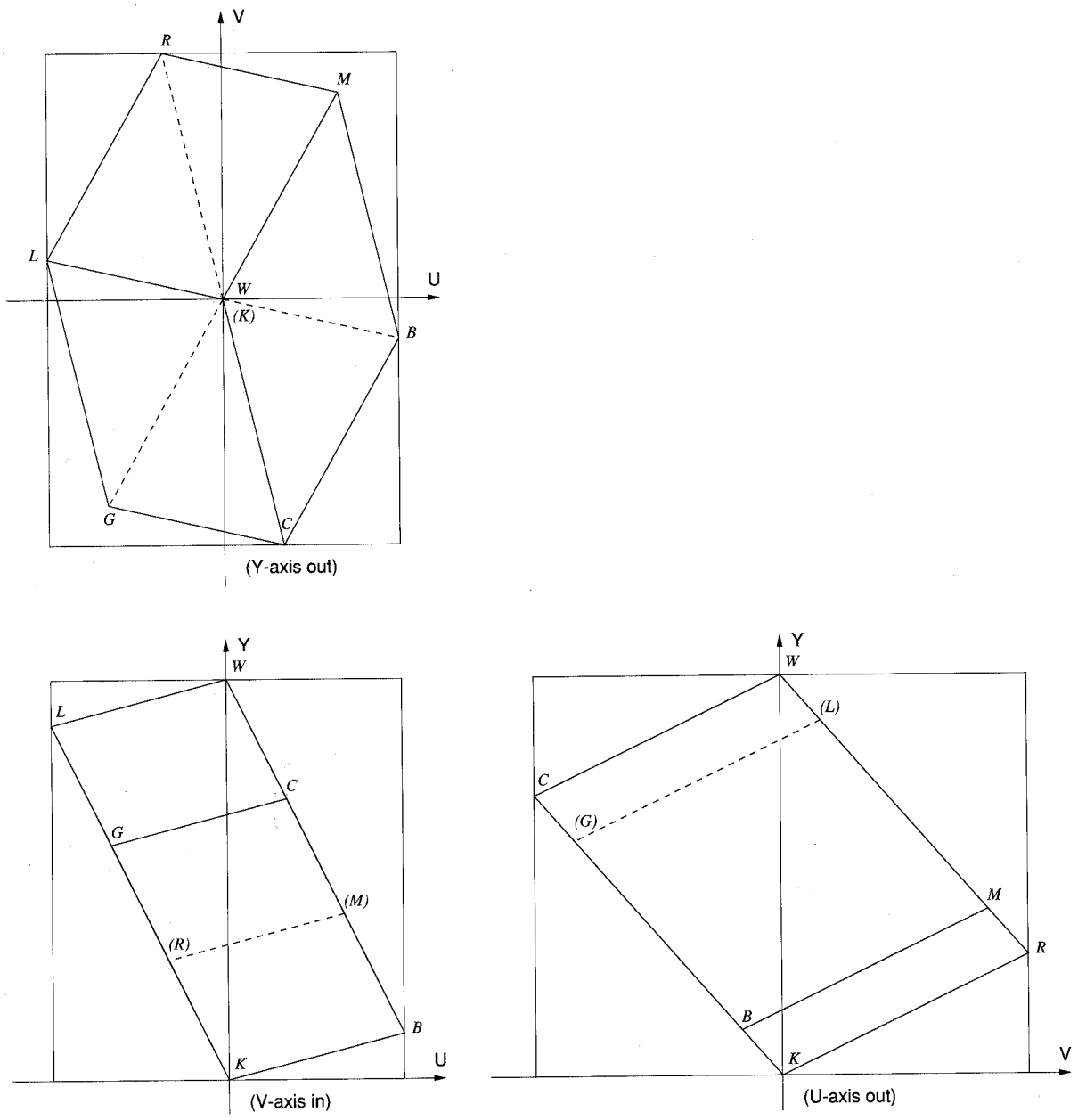


Figure 5 Feasible RGB Values in the YUV Color Space

parallelepiped of "feasible" RGB space. Feasible RGB points are those that are nonnegative and are not greater than the maximum supported value. For reference, the corners of the RGB parallelepiped are labeled black (K), white (W), red (R), green (G), blue (B), cyan (C), magenta (M), and yellow (L). RGB and YUV values are related linearly and can be interconverted by means of a 3-by-3 matrix multiply.

In the United States video broadcast system, the chrominance plane (i.e., the U-V plane in Figure 5) is rotated 33 degrees by introducing a phase in the quadrature modulation of the chrominance signal. The resulting rotated chrominance signals are renamed I and Q (for inphase and quadrature), but the unmodulated color space is still YUV.

Figure 6 shows the back end of a rendering system that uses dithering as a quantization step prior to color space conversion. A serendipitous consequence of dithering is that color space conversion can be achieved by means of table look-up. The collective address formed by the dithered Y, U, and V values is small enough to require a reasonably sized color mapping LUT. There are two advantages to this approach. First, a costly dematrixing operation is not required, and second, infeasible RGB values can be intelligently mapped back to feasible space off-line during the generation of the color mapping LUT.

This second advantage is an important one, because 77 percent of the valid YUV coordinates are in invalid RGB space, i.e., the space around the RGB parallelepiped in Figure 5. Color adjustments such as increasing the brightness or saturation can push otherwise valid RGB values into infeasible space. In alternative systems that perform color conversion by dematrixing, out-of-bounds RGB values are simply truncated. This operation effectively maps colors back to feasible RGB space along lines perpendicular to a parallelepiped surface illustrated in Figure 5, which can change the color in an undesirable way. The use of a color mapping LUT avoids these problems.

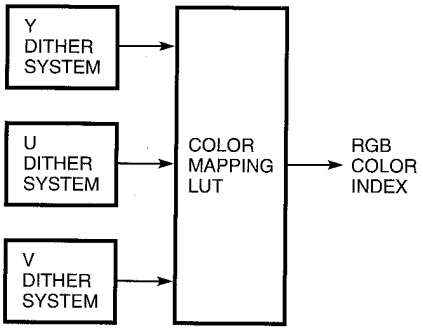


Figure 6 System for Dithering Three-color Components and Color Mapping the Collective Result

Summary

Video is becoming an increasingly important data type for desktop systems. This is especially true as distinctions between computing, consumer electronics, and communications continue to blur. While many factors contribute to the impression one has of the value of a product that displays information, the way the images look can make the biggest difference. This paper focuses on rendering system designs that are fast, low cost, produce good-quality video, and are conducive to hardware or software implementation.

References

1. *Special Issue on Digital Multimedia Systems, Communications of the ACM,* vol. 34, no. 1 (April 1991).

2. B. Neidecker-Lutz and R. Ulichney, "Software Motion Pictures," *Digital Technical Journal,* vol. 5, no. 2 (Spring 1993, this issue): 19–27.

3. W. Schreiber and D. Troxel, "Transformation between Continuous and Discrete Representation of Images: A Perceptual Approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI),* vol. PAMI-7, no. 2 (1985): 178–186.

4. R. Ulichney, *Digital Halftoning* (Cambridge, MA: The MIT Press, 1987).

5. J. Bresenham, "Algorithm for Computer Control of a Digital Plotter," *IBM Systems Journal,* vol. 4, no. 1 (1965): 25–30.

6. E. Glazer, "Fast Bitonal to Grayscale Image Scaling," DEC-TR-505 (Maynard, MA: Digital Equipment Corporation, June 1987).

7. R. Ulichney, "Bresenham-style Scaling," *Proceedings of the IS&T Annual Conference* (Cambridge, MA, 1993): 101–103.

8. P. Heckbert, "Color Image Quantization for Frame Buffer Display," *Computer Graphics AMC SIGGRAPH '82 Conference Proceedings,* vol. 16, no. 3 (1982): 297–307.

9. S. Wan, K. Wong, and P. Prusinkiewicz, "An Algorithm for Multidimensional Data Clustering," *ACM Transactions on Mathematical Software,* vol. 14, no. 2 (1988): 153–162.

10. C. Sigel, R. Abruzzi, and J. Munson, "Chromatic Subsampling for Display of Color Images," *Optical Society of America Topical Meeting on Applied Vision,* 1989 Technical Digest Series, vol. 16 (1989): 158–161.

11. A. Luther, *Digital Video in the PC Environment* (New York, NY: Intertext Publications, McGraw-Hill, 1989): 193–194.

12. L. Glass, "Digital Video Interactive," *Byte* (May 1989): 284.

13. P. Roetling, "Binary Approximation of Continuous-tone Images," *Photographic Science and Engineering,* vol. 21 (1977): 60–65.

14. J. Stoffel and J. Moreland, "A Survey of Electronic Techniques for Pictorial Reproduction," *IEEE Transactions on Communications,* vol. 29 (1981): 1898–1925.

15. J. Jarvis, C. Judice, and W. Ninke, "A Survey of Techniques for the Display of Continuous-tone Pictures on Bilevel Displays," *Computer Graphics and Image Processing,* vol. 5 (1976): 13–40.

16. W. Goodall, "Television by Pulse Code Modulation," *Bell Systems Technical Journal,* vol. 30 (1951): 33–49.

17. B. Bayer, "An Optimum Method for Two Level Rendition of Continuous-tone Pictures, *Proceedings of the IEEE International Conference on Communications, Conference Record* (1973): (26-11)-(26-15).

18. R. Ulichney, "Frequency Analysis of Ordered Dither," *Proceedings of the Society of Photo-optical Instrumentation Engineers (SPIE),* vol. 1079 (1989): 361–373.

19. R. Ulichney, "The Void-and-cluster Method for Dither Array Generation," *The Society for Imaging Science and Technology/Symposium on Electronic Imaging Science and Technology (IS&T/SPIE)* (February 1993).