Larry D. Seiler
Robert A. Ulichney

Integrating Video Rendering into Graphics Accelerator Chips

The fusion of multimedia and traditional computer graphics has long been predicted but has been slow to happen. The delay is due to many factors, including their dramatically different data type and bandwidth requirements. Digital has designed a pair of related graphics accelerator chips that integrate video rendering primitives with two-dimensional and three-dimensional synthetic graphics primitives. The chips perform one-dimensional filtering and scaling on either YUV or RGB source data. One implementation dithers YUV source data down to 256 colors. The other converts YUV to 24-bit RGB, which is then optionally dithered. Both chips leave image decompression to the CPU. The result is significantly faster frame rates at higher video quality, especially for displaying enlarged images. The paper compares the implementation cost of various design alternatives and presents performance comparisons with software image rendering.

For years, the computer industry confidently predicted that ubiquitous, integrated multimedia computing was just around the corner. After a number of delays, this computing environment is finally a reality. It is now possible to buy personal computers (PCs) and workstations that combine audio processing with real-time display and manipulation of video or other sampled data, though usually with significant limitations.

For the most part, the industry has followed one of two paths to achieve real-time video processing. On one path, video features are implemented almost entirely in software. When applied to the display of moving images, this approach typically results in a combination of low resolution, slow update times, and small images.

The alternative has been to achieve good video image display performance by adding a separate video hardware option to a PC. Image display is integrated in the box and on the screen but is distinct from the hardware that implements traditional synthetic graphics. Frequently, this design forces performance compromises, for example, by limiting the number of video images that can appear at the same time or by limiting the interaction of images with the window system.

Recently, two key enabling technologies have combined to make a better solution possible. Advances in silicon technology enable low-cost graphics controller chips to be designed with a significant number of gates dedicated to supporting multimedia features. In addition, the peripheral component interconnect (PCI) bus provides high-bandwidth, peer-to-peer communication between the CPU, the main memory, and option cards. Peak bandwidth on the standard 32-bitPCI bus is 133 megabytes per second (MB/s), and higher-performance versions are also available. Good PCI implementations can transfer sequential data at 80 to 100 MB/s. Equally important, the PCI bus allows multimedia solutions to be incrementally built up from a software-only implementation through various levels of hardware support. The *PCI Multimedia Design Guide* describes this incremental approach and also provides standards for latency and video data formats.[1]

This paper describes a Digital engineering project whose goal was to combine video rendering features and traditional synthetic graphics into a unified graphics chip, yielding high-quality, real-time image display as part of the base graphics option at minimal extra cost. This project resulted in two chip implementations, each with its own variation of the same basic design. The TGA2 chip was designed in the Worksystems Group for use in Digital's PowerStorm 3D30 and PowerStorm 4D20 graphics options. The Dagger chip (DECchip 21130) was designed in the Silicon Engineering Group to match the needs of thePC market. The TGA2 and Dagger chips are PCI bus masters and can accept video data from either the hostCPU or other video hardware on the PCI bus.

The basic block diagram of the two chips is illustrated in Figure 1. PCI commands are interpreted as either direct memory access (DMA) requests or drawing commands, which the pixel engine block converts to frame buffer read and write operations. Alternately, PCI commands can directly access the frame buffer or the video graphics array (VGA) and RAMDAC logic. In the Dagger chip, the VGA and RAMDAC logic is on-chip; in the TGA2 chip, this logic is implemented off-chip. Most of the video rendering logic is contained in the pixel engine block; the command interpreter and DMA engine blocks require some additional logic to support video rendering.

The following sections describe the capabilities, costs, and trade-offs of the video rendering feature set as implemented in the Dagger and TGA2 graphics chips.

Defining a Low-level Video Rendering Feature Set

The key question when integrating multimedia into a traditional synthetic graphics chip is which features should be implemented in hardware and which should be left in software. A cost-effective design cannot include enough gates to implement every feature of interest. In addition, time-to-market concerns do not allow all features to be designed into the hardware. Therefore, it is essential for designers to define the primary trade-off between features that can be easily and effectively implemented in hardware and those that can be more easily implemented in software without compromising performance.

For the Dagger and TGA2 graphics chips, our basic decision was to leave image compression and decompression in software and put all pixel processing operations into hardware. This approach lets software do what it does best, which is perform complex control of relatively small amounts of data. It also lets hardware do what it does best, which is process large amounts of data where the control is relatively simple and is independent of the data. Specifically, in these two graphics chips, image scaling, filtering, and pixel format conversions are all performed in hardware.

Performing the scaling in hardware greatly reduces the amount of data that the software must process and that must be transmitted over the PCI bus. For example, a 320-by-240-pixel image represented with 16-bit pixels requires just 150K bytes. Even at 30 frames per second (fps), transmitting an image of this size consumes about 5 percent of the available bandwidth of a good PCI bus implementation. This data could be displayed as a 1,280 by 960 array of 32-bit pixels for display, which would use more than 80 percent of the PCI bus bandwidth, if the scaling and pixel format conversion occurs in software.

One data-intensive operation that we chose not to implement in hardware is video input. Designers will need to revisit this decision with each new generation

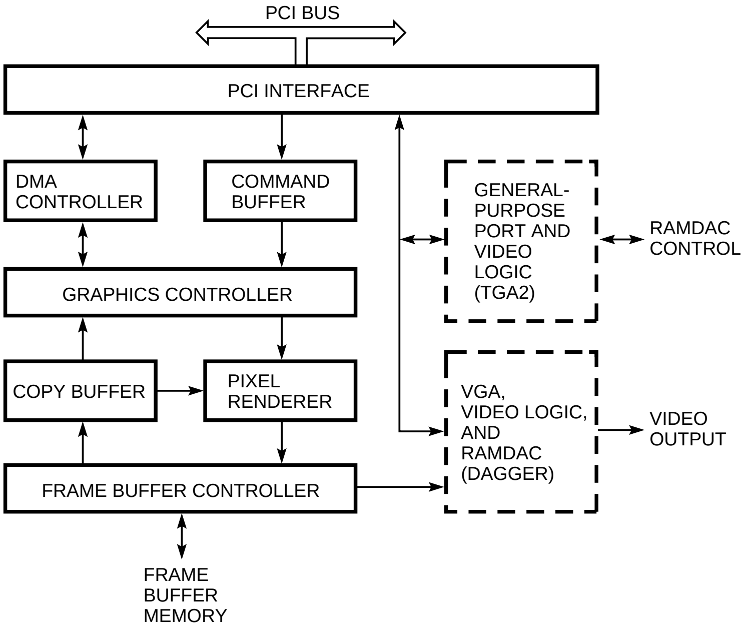

Figure 1
Dagger and TGA2 Chip Structure

of graphics chips. For the current generation, we decided to require the use of a separate video input card for the subset of systems that require video capture. We decided not to include video capture support in the Dagger and TGA2 chips for two basic reasons. First, current application-specific integrated circuit (ASIC) technology would have allowed only a partial solution. We could have put a video input port in hardware but could not have supported the complex operations needed for image compression.

The second reason stems from a market issue. Video display is rapidly becoming ubiquitous, just as mice and multiwindow displays have become commonplace for interacting with PCs and workstations. It is now practical to support high-quality, real-time video display in the base graphics chip. However, the market for video input stations is still much smaller than the market for video display stations. When the size of the video input station market is large enough, and the cost of integrating video input is small enough, support for video input should be added to the base graphics chip.

Video Rendering Pipeline

This section describes the stages of video rendering that are implemented in the Dagger and TGA2 graphics chips. These stages are pixel preprocessing, scaling and filtering, dithering, and color conversion. In some cases, such as scaling and filtering, the two implementations are practically identical. In others, such as color conversion, dramatically different implementations are used to address the differences in requirements for the two chips.

Pixel Preprocessing

The first stage in the pipeline inputs pixel data and converts it into a standard form to be used by the rest of the pipeline. This involves both converting input pixels to a standard format and pretranslating pixel values or color component values. The Dagger and TGA2 chips use DMA over the PCI bus to read packed arrays of pixels from memory.

Pixel Format Conversion Multimedia images are typically represented in YUV format, where the *Y* channel specifies luminance and the *U* and *V* channels represent chrominance. After the CPU has decompressed the source image into arrays of *Y, U,* and *V* pixel values, this data is transmitted to the graphics chip in one of a number of standard formats. Alternately, images may be specified as red/green/blue (RGB) triples instead of YUV triples, or as a single index value that specifies a color from a color map random-access memory (RAM) in the video logic. The *PCI Multimedia Design Guide* specifies many standard pixel formats.[1]

Figure 2 shows some of the input pixel formats that are supported in the Dagger and TGA2 graphics chips. The YUV formats on the left allocate 8 bits for each channel. The upper format of the four uses 32 bits per YUV pixel and is called YUV-4:4:4+α.[1] The alpha field is optional and is not used in the Dagger and TGA2 chips. Alpha values are used for blending operations with partially transparent pixels. An alpha value of zero represents a fully transparent pixel, and the maximum value represents a fully opaque pixel.

The remaining three YUV formats specify a separate *Y* value per pixel but subsample the *U* and *V* values so that a pair of pixels shares the same *U* and *V* values. Most YUV compression schemes subsample the chrominance channels, so this approach does not represent any loss of data from the decompressed image. Since the human visual system is more sensitive to changes in luminance than to changes in chrominance, for natural images, *U* and *V* can be subsampled with little loss of image quality.

The three 16-bit YUV formats represent the most common orderings for chrominance-subsampled YUV values. The little-endian and gib-endian orderings are called YUV-4:2:2.[1] The little-endian ordering is the order that is typically produced on the PCI bus

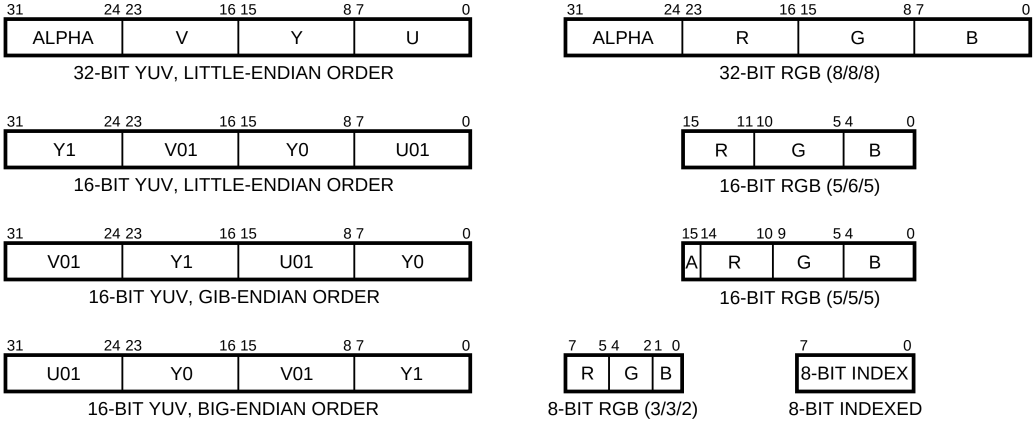

Figure 2
YUV and RGB Pixel Formats in the Dagger and TGA2 Chips

by a little-endian machine. The gib-endian ordering is produced on the PCI bus by a big-endian machine that converts its data to little-endian order, as required for transfer across the PCI bus. That operation preserves byte order for 8-bit and 32-bit data types but not for 16-bit data types like this one. Finally, the big-endian byte ordering is used by some video rendering software and hardware options.

The RGB formats on the right side of Figure 2 allocate varying numbers of bits to the red, green, and blue color channels to produce 8-bit to 32-bit pixels. To achieve acceptable appearance, 8-bit RGB requires high-quality dithering, such as that provided by the AccuVideo dithering technology contained in the Dagger and TGA2 chips and described later in this section. Thirty-two-bit RGB has an optional alpha channel that is not used in the Dagger and TGA2 chips. Some hardware uses the field for control bits or overlay planes instead of for the alpha value. Two different 16-bit RGB formats are common. One format provides 5 bits per color channel and a single alpha bit that indicates transparent or opaque. The other format provides an extra bit for the green channel, since the eye is more sensitive to green than to red or blue.

Finally, 8-bit indexed format is shown at the bottom of Figure 2. This format is simply an 8-bit value that represents an index into a color map. Dagger has an integral color map and digital-to-analog converter, whereas TGA2 requires an external RAMDAC chip to provide its color map. The 8-bit indexed format can represent an indexed range of values or simply a collection of independent values, depending on the needs of the application. In the Dagger and TGA2 chips, the 8-bit indexed format is processed by being passed through the *Y* channel.

Once in the pipeline, the pixels are converted to a standard format consisting of three 8-bit values per pixel. The three values represent RGB or YUV components, depending on the original pixel format. If the original field contains fewer than 8 bits, for example, in the 8-bit RGB format, then the available bits are replicated. Figure 3 shows the expansion of RGB pixels to 8/8/8 RGB format. Replicating the available bits to fill low-order bit positions is preferable to filling the low-order bits with zeros, since replication stretches out the original range of values to include both the lowest and highest values in the 8-bit range, with roughly equal steps between them.

Adjust Look-up Table In the TGA2 chip, a 256-entry look-up table (LUT) may be used during pixel preprocessing. Figure 7 (discussed in the section Color Conversion Algorithms) shows this table, called the adjust LUT, in the TGA2 pipeline. This table supports two different data conversions: luminance adjustment and color index conversion. The adjust LUT is not available in the Dagger chip because it requires too many gates to meet the chip cost goal for Dagger.

Luminance adjustment is used with YUV pixel formats. When this feature is selected, the 8-bit *Y* value from the input pixel is used as an index into the adjust LUT. The 8-bit value read from the table is used as *Y* in the next pipeline stage. Proper programming of the table allows arbitrary luminance adjustment functions to be performed on the input *Y* value; brightness and contrast control are typically provided through this mechanism. Standards for digitally encoding video specify limited ranges for the *Y, U,* and *V* values, largely to prevent analog noise from creating out-of-range values.[2] A particularly important use of this luminance-adjust feature is correcting the poor contrast that would otherwise result from this range limitation. In this case, the adjust LUT may be used to remap the *Y* values to cover the full range of values from 0 to 255.

Another desirable feature is chrominance adjustment, under which the *U* and *V* values are also arbitrarily remapped. The J300 provides this feature; however, TGA2 does not, for two reasons.[3] First, chrominance adjustment is required less often than luminance adjustment and can be emulated in software when the feature is required. Second, chrominance adjustment consumes a significant amount of chip area—either 2K or 4K bits of memory, depending on whether *U* and *V* use the same table or different tables. In this generation of graphics chips, the feature could not be justified in the TGA2 chip. The Dagger chip, which was

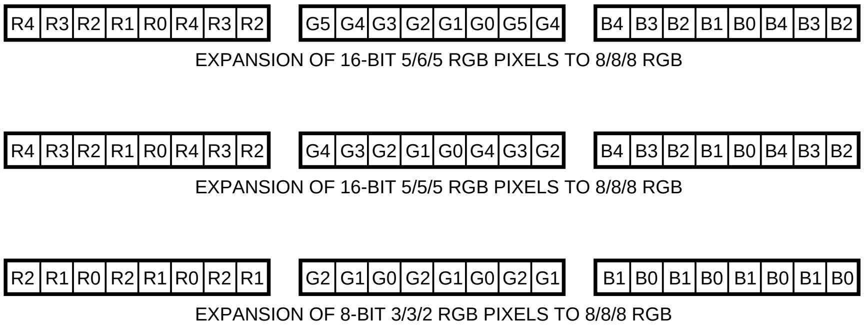

Figure 3
Expanding RGB Pixels to 8/8/8 RGB Format

intended for lower-cost systems, includes neither chrominance nor luminance adjust LUTs.

The other use for the adjust LUT in the TGA2 chip is for color index conversion. This operation can be performed when the input pixel format is 8 bits wide. In this case, the 8-bit input pixel is used as an index into the table. The resulting value is used as the *Y*-channel value in the rest of the pipeline, and the *U* and *V* channels are ignored. Later in the pipeline, the color conversion stage is skipped, and the *Y*-channel value is used directly as the resulting 8-bit pixel value.

Color index conversion is an operation that is particularly desirable when using the Windows NT operating system. Typically, 8-bit screen pixels are converted to displayed colors by means of a colorLUT in the back-end video logic. Under the X Window System graphical windowing environment, the mapping between an index and its color can be changed only by the application. Under the Windows NT operating system, however, the mappings may change dynamically. Therefore, an application that has stored an image as 8-bit index values will need to remap those index values before copying it to the screen. This conversion can be done in software, but it is faster and simpler to use the adjust LUT in the TGA2 chip to perform the remapping.

Scaling and Filtering

In the next stage in the rendering pipeline, the chip performs scaling and filtering. The Dagger and TGA2 chips support one-dimensional (1-D) scaling and filtering in hardware. Limiting the chips to 1-D filtering significantly simplifies the chip logic, since no line buffers are needed. Somewhat higher-quality images can be achieved using two-dimensional (2-D) filtering, but the difference is not significant. This difference is further reduced by the AccuVideo dithering algorithm that is implemented by the Dagger and TGA2 chips. Two-dimensional smoothing filters can be supported with added software processing, if required.

Bresenham-style Scaling Image scaling in the Dagger and TGA2 chips uses pixel replication but is not limited to integer multiples. Instead, images can be scaled from any integral source width to any integral destination width. Scaling is implemented through an adaptation of the Bresenham line-drawing algorithm. A complete description of this Bresenham-style scaling algorithm appears in "Bresenham-style Scaling"; the following paragraphs provide an outline of the algorithm, which is the same scaling algorithm used in the J300 family of adapters.[3,4]

The Bresenham scaling algorithm works like the Bresenham line-drawing algorithm. Suppose we are drawing a line from (0, 0) to (10, 6), so that $dx = 10$ and $dy = 6$. This is an *X*-major line; that is, the line is longer in the *X* dimension than in the *Y* dimension. The Bresenham algorithm draws this vector by initializing an error term and then incrementing it *dx* times, in this example, 10 times. Each time the algorithm increments the term, a pixel is drawn. The sign of the error term determines whether to find the next pixel position by stepping to the right (incrementing the *X* position) or by stepping diagonally (incrementing both *X* and *Y*). The error term is incremented in such a way that as the *X* position is incremented 10 times, the *Y* position is incremented 6 times, thus drawing the desired vector.

For Bresenham scaling, *dx* represents the width of the source image, and *dy* represents the width of the destination image on the screen. When reducing the size of the source image, *dx* is greater than *dy* and the error terms and increments are set up in the same way as the *X*-major Bresenham line drawing, as described in the previous paragraph. One source pixel is processed each time the error term is incremented. When Bresenham's line algorithm indicates a step in the *X* dimension only, the source pixel is skipped. When the algorithm indicates a step in both the *X* and the *Y* dimensions, the source pixel is written to the destination. As a result, exactly *dx* source pixels are processed, and exactly *dy* of them are drawn to the screen.

Enlarging an image works in a similar fashion. For example, consider a source image that is narrower than the destination image, that is, *dx* is less than *dy.* This is equivalent to drawing a *Y*-major Bresenham line in which the error term is incremented *dy* times and the *X* dimension is incremented *dx* times. The scaling algorithm draws a source pixel to the destination at each step. If the line-drawing algorithm increments only in the *Y* dimension, it repeats the current pixel. If the line-drawing algorithm increments in both the *X* and the *Y* dimensions, it steps to and displays the next source pixel. Consequently, the *dx* source pixels are replicated to yield *dy* destination pixels, thus enlarging the image.

The Bresenham line-drawing algorithm has two nice properties that are shared by the Bresenham scaling algorithm. First, it requires no divisions to compute the error increments. Second, it produces lines that are as smooth as possible, given the pixel grid. That is, for an *X*-major line, each of the *dx* pixels has a *Y* position that is the closest pixel to the intersection of its *X* position with the real vector. Similarly, the Bresenham scaling algorithm selects pixels that have the most even spacing possible, given the pixel grid.

Just as lines can be drawn from left to right or from right to left, images can be drawn in either direction. An image drawn in one direction is the mirror image of the image drawn in the other direction. Mirror imaging is sometimes used in teleconferencing, so that users can look at themselves the way they normally see themselves. Similarly, images can be turned upside down by simply drawing to the display from bottom to top instead of from top to bottom.

Scaling in the *Y* dimension is performed similarly to *X*-dimension scaling. On the TGA2 chip, scaling is performed in software instead of in hardware: the software increments an error term to decide whether to skip lines (for reducing) or repeat lines (for enlarging). This is acceptable because the CPU has plenty of spare cycles to perform the scaling computations while the algorithm draws the preceding line. The Dagger chip supports *Y*-dimension scaling in hardware to reduce the number of commands that are needed to scale an image.

Smoothing and Sharpening Filters Like the J300, the Dagger and TGA2 chips provide both smoothing and sharpening filters. Table 1 shows the available filters. All are three-tap filters that are inexpensive to implement in hardware. The smoothing filters are used to improve the quality of scaled images. The sharpening filters provide edge enhancement. The two filters marked with asterisks (*) are available only on the TGA2 chip. The others are available on both the Dagger and the TGA2 chips.

The three rows of Table 1 show three levels of smoothing and sharpening filters that can be applied. The degree of smoothing and sharpening may be selected separately. The first row shows the identify filter. This is selected to disable smoothing or sharpening. The second and third rows show three-tap filters that perform a moderate and an aggressive degree of smoothing or sharpening.

Note that when using the aggressive smoothing filter, the center element does not contribute to the result. This filter is intended for postenlargement smoothing when the scale factor is large. Since enlargement is performed by replicating some of the pixels, the center of any span of three pixels will be identical to one of its neighbors when scaling up by a factor of two or more. As a result, the center pixel affects the resulting image, since it is replicated either to the left or to the right. The (1/2, 0, 1/2) filter affords the greatest degree of smoothing that can be achieved with a three-tap filter.

These filter functions are simple to implement in hardware. The implementation requires storing only the two preceding pixels and performing from one to three addition or subtraction operations. The sharpening filters require an additional clamping step to ensure that the result is in the range 0 to 1. Better filtering functions could be obtained by using five taps instead of three taps but only by significantly increasing the logic required for filtering.

Table 1
Smoothing and Sharpening Filters

Smoothing Filter	Degree of Filtering	Sharpening Filter
(0, 1, 0)	Unfiltered	(0, 1, 0)
(1/4, 1/2, 1/4)*	Moderate	(−1/2, 2, −1/2)
(1/2, 0, 1/2)	Aggressive	(−1, 3, −1)*

* Available only on the TGA2 chip

Pre- and Postfiltering The order in which filters are applied depends on whether the image is being enlarged or reduced. When reducing an image, the Bresenham scaling algorithm eliminates pixels from the source image. This can result in severe aliasing artifacts unless a smoothing filter is applied before scaling. The smoothing filter spreads out the contribution of each source pixel to adjacent source pixels.

When enlarging an image, the smoothing filter is applied after scaling. This smoothes out the edges between replicated blocks of pixels. The smoothing filters eliminate the block effect entirely when enlarging up to two times the source image size. The AccuVideo dithering algorithm also contributes to smoothing out the edges between blocks. Another way to smooth out the edges is to use higher-order interpolation to find destination pixel values. Such methods require more logic and do not necessarily produce a better-looking result, particularly for modest scale factors.

If sharpening or edge enhancement is desired, a sharpening filter is used in addition to whatever smoothing filter is selected. For reducing an image, the sharpening filter is applied after scaling—sharpening an image before reducing its size would only exaggerate aliasing effects. For enlarging an image, the sharpening filter is applied before scaling—sharpening an image after enlarging its size would only amplify the edges between blocks. As a result, when both sharpening and smoothing filters are used, one is applied before scaling and the other is applied after scaling.

AccuVideo Dithering Algorithm

AccuVideo dithering technology is Digital's proprietary high-quality, highly efficient method of rendering video with an arbitrary number of available colors. Included is YUV-to-RGB conversion, if necessary, with careful out-of-bounds color mapping. The general algorithm is described in two other papers in this issue of the *Journal,* which discuss the implementation of the J300 video adapter and software-only video players.[3,5] In the chips described in this paper, we simplified the general implementation of the AccuVideo technology by setting constraints on the number of available colors.

Review of the Basic Algorithm The development of the general mean-preserving multilevel dithering algorithm is presented in "Video Rendering," which appears in an earlier issue of the *Journal.*[6] Figure 4 illustrates the theoretical development of the fundamental algorithm for dithering a simple component of a color image. As stated in the earlier paper,

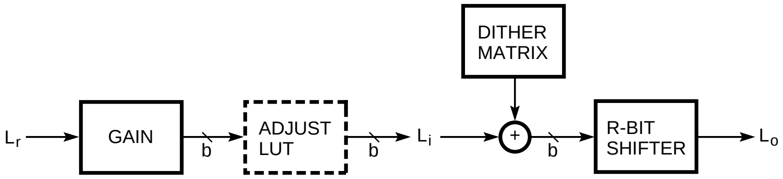

Figure 4
Multilevel Dithering Algorithm Used in the J300, with the Gain Function Separated from the Adjust LUT

a mean-preserving dithered output level L_o can be produced by quantizing the sum of an element from a normalized dither array and an input level L_i by simply shifting the sum to the right by R bits. This simplified quantizer, that is, a quantizer with step size $\Delta_Q = 2^R$, is possible only if the range of input to the adder L_i, or the number of input levels N_i, is properly scaled by a gain G. In the J300 and software-only implementations, G is included in an adjust LUT. In Figure 4, we explicitly separate G from the adjust LUT. The adjust LUT is optionally used to control characteristics such as contrast, brightness, and saturation.

The components of this dithering system can be designed by specifying three parameters:

1. N_r, the number of raw input levels of the given color component
2. N_o, the number of desired output levels
3. b, the width of the adder in bits, and the number of bits used to represent the input levels

Using the results from the multilevel dithering algorithm, the number of bits to be right-shifted is

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

and the gain is

$$G = \frac{N_i - 1}{N_r - 1},$$

where

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

The effect of the gain is multiplicative. That is, $L_i = L_r \times G$, where L_r is the raw input level. In the absence of an adjust LUT, this multiplication must be explicitly performed.

Simplified Implementation of Gain In the above summary of the basic dithering algorithm, the values of N_r and N_o can be any integer, where $N_r > N_o$. Consider the important special case of restricting these values to be powers of two. Introducing the three integers p, q, and z, we specify that $N_r = 2^p$, $N_o = 2^q$, and $b = p + z$, where z is the number of additional bits used to represent L_i over L_r. $z > 0$ guarantees that $N_i > N_r$, thus ensuring that all the raw input levels will be distinguished by the dithering system. $z = 0$ causes $N_i < N_r$. This situation results in some loss of resolution of raw input levels, because, in all cases, the number of perceived output levels from the dithering system will be at most N_i.

Using this information and the expressions of R and G, it is straightforward to show that $R = p - q + z$, and

$$N_i = (2^q - 1)2^R + 1.$$

Further,

$$G = \frac{((2^q - 1)2^R + 1) - 1}{2^p - 1} = \frac{(2^q - 1)2^p}{(2^p - 1)2^{(q-z)}}.$$

A key approximation made at this point is

$$\frac{2^p}{2^p - 1} \approx 1.$$

Note that this approximation becomes better as the number of bits, p, in the raw input increases.

An approximate gain thus simplifies to

$$\hat{G} = \frac{(2^q - 1)}{2^{(q-z)}} = 2^z - \frac{1}{2^{(q-z)}}.$$

With this value of $\hat{G}$, the resulting modified input levels will be proportionally less than ideal by a factor of

$$\frac{\hat{G}}{G} = \frac{2^p - 1}{2^p}.$$

The fact that this error is negative guarantees that overflow will never occur in the multilevel dithering system. Therefore, a truncation step is not needed in the implementation. Figure 5 illustrates the implementation of $\hat{G}$, which consists of the subtraction of a $(q - z)$-bit right shift of L_r from a z-bit left shift of L_r. This simple "multiplier" is what is implemented in Dagger, TGA2, and the ZLX family of graphics accelerators, where the power-of-two constraint on the output levels is made.

Consider, for example, the case where $p = 8$ ($N_r = 256$), $q = 3$ ($N_o = 8$), and $z = 1$. From the equations just presented, $R = 6$, $b = 9$, and $N_i = 449$. Although our approximation for the gain, $\hat{G} = (2 - 1/4) = 1.75$,

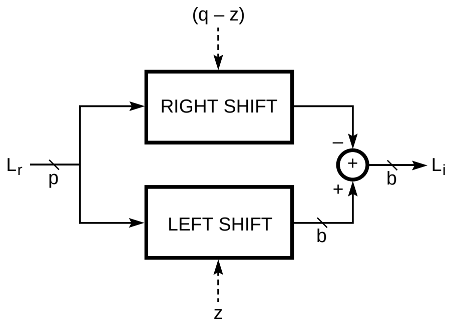

Figure 5
Parallel-shifter Implementation of the Gain Function

is not equal to the ideal gain, $G = 448/255 \approx 1.757$, the ratio $\hat{G}/G \approx 0.996$ is so close to unity that any resulting differences in output are indistinguishable.

Shared Dither Matrix Another simplification can be made by having all the color components in the rendering system share the same dither matrix. As defined in "Video Rendering," a dither template is an array of N_t unique elements, with values $T \in \{0, 1, \ldots, (N_t - 1)\}$.[6] These elements are normalized to establish the dither matrix element d for each location $[x, y]$ as follows:

$$d[x,y] = \text{int}\left\{\frac{2^R}{N_t}\left(T[x,y]+\frac{1}{2}\right)\right\}.$$

For any real number A and any positive integer K, the following is always true:

$$\text{int}\left\{\frac{A}{K}\right\} = \text{int}\left\{\frac{\text{int}\,A}{K}\right\}.$$

If, for each color component, N_o is a power of two, we can exploit this fact by storing only a single dither matrix designed for the smallest value of N_o. Specifically, this would be $N_o = 2^{(b - R_m)}$, where b is the width in bits of the adder and R_m is the largest value of R in the system. For the other larger number of output levels $N_o' = 2^{(b - R')}$ with smaller values of R', normalized dither matrix values $d'[x, y]$ can easily be derived by a simple right shift by $(R_m - R')$ bits of the stored dither matrix, as shown in the following equation:

$$d'[x,y] = \text{int}\left\{\frac{d[x,y]}{2^{R_m - R'}}\right\}.$$

Since our dither matrices are typically 32 by 32 in size, the hardware savings in storing only one matrix is significant. Also, the stored values can be read-only memory (ROM) instead of the more costly RAM. Typically, RAM requires up to eight times the area of ROM in either gate array or custom implementations.

Color Conversion Algorithms

The result of the preceding pipeline stages is three 8-bit values that represent either RGB or YUV color channels. If this format is to be written to the frame buffer, then no further processing is necessary. If a different destination format is specified, then Dagger and TGA2 must perform a color format conversion. Both chips use the same algorithm to dither RGB values down to a smaller number of bits per color channel. Both chips allow writing YUV pixels to the frame buffer, although TGA2 allows the writing of only the 32-bit YUV format. Finally, both chips can convert YUV pixels into the RGB color space, but they use markedly different algorithms to perform this conversion.

Although YUV pixels can be written to the frame buffer in both Dagger and (to a more limited extent) TGA2, neither chip supports displaying YUV pixels to the screen. YUV pixels may be stored only in the off-screen portion of the frame buffer as intermediate values for further processing. This is because it is far more efficient to convert YUV to RGB in the rendering stage than to perform the conversion in the back-end video logic. At the rendering stage, it need only be done at the image update rate of up to 30 fps. If performed in the back-end video logic, the YUV-to-RGB conversion must also be performed at the screen update rate of up to 76 fps. This extra, higher-speed logic may be justified if preconverting YUV to RGB noticeably reduces the image quality. Given the AccuVideo dithering algorithm, however, postconversion is not necessary.

RGB-to-RGB Color Conversion Even if both the source and the destination pixel formats represent RGB color channels, it may still be necessary to perform a bit-depth conversion. Input pixels are expanded out to 8 bits per color channel for processing through the video rendering pipeline. Destination pixels may have 8, 15, 16, or 24 bits for RGB and so may need to be dithered down to a smaller number of bits per pixel. TGA2 also supports 12-bit RGB, as described later in this section.

Dagger and TGA2 differ somewhat in the specific formats that they support. Dagger allows writes to the frame buffer of 3/3/2, 5/5/5, 5/6/5, and 8/8/8 RGB pixel formats. TGA2 supports all these as source pixels but does not allow writes of 5/5/5 and 5/6/5 RGB, because TGA2 does not support 16-bit pixels in the frame buffer. Dagger supports 16-bit pixels because they are very common in the PC industry. In the workstation industry, however, which is TGA2's market, 16-bit pixels are almost unknown. As the Windows NT operating system gains in popularity, this situation is likely to change.

Instead of supporting 16-bit pixels, TGA2 allows writes to the frame buffer of 4/4/4 RGB pixels, with 16 possible shades for each of the red, green, and blue color channels. This is a standard pixel format for workstation graphics, since it allows two RGB buffers to be stored in the space of a 24-bit, 8/8/8 RGB pixel. This in turn allows double buffering, in which one image is drawn while the other image is displayed. Double buffering is essential for animation applications on large screens, since the rendering logic generally cannot repaint the screen fast enough to avoid flicker effects.

YUV-to-RGB Color Conversion on the Dagger Chip

The key design focus for the Dagger chip was to support low-cost graphics options with the highest possible performance and display quality. As a result, although Dagger supports up to 32 bits per pixel, its design center is for 8-bit-per-pixel displays. Therefore, the algorithm that Dagger uses for converting YUV to RGB produces the best possible results given a limit of just 256 resultant colors.

The resulting dithering system design is shown in Figure 6. Note that the same system is used to dither both RGB data and YUV data. Because the number of output levels for each component is always a power of two, we can use the simple gain circuit of Figure 5 and share the same dither matrix by right-shifting its contents, as derived in the last section. In hardware, this shifting simply requires a multiplexer to select the most significant bits of the data. The dither matrix is 7 bits wide to support dithering down to 2-bit blue values in 3/3/2 RGB, but only 6 dither matrix bits are used for 3-bit output, and only 5 bits are used for 4-bit output.

YUV data is always dithered to 4 bits of *Y* and 3 bits each of *U* and *V*. An additional bit is provided for the *Y* channel because the eye is more sensitive to changes of intensity than to changes of color. These 10 bits are input to a color convert LUT, which is implemented as a ROM. Its contents are generated by an algorithm with some out-of-bounds mapping.[5,7] Approximately three-fourths of the possible combinations of YUV values are outside the range of colors that can be specified in the RGB color space. In these cases, the color convert LUT ROM produces an RGB value that has the same luminance but a less saturated color.

The color convert LUT ROM represents these 256 colors as an 8-bit index that is stored in the frame buffer. One additional bit per pixel in off-screen memory specifies which pixels result from YUV conversion and which are used by other applications. When pixels are read from the frame buffer for display to the screen, Dagger's internal RAMDAC reads that additional bit per pixel to decide whether to map each byte through a standard 256-entry color map or through a ROM that is loaded with the 256 colors selected in the color convert LUT ROM. As a result, Dagger allows selection of the best 256 colors for YUV-to-RGB conversion, in addition to allowing color-mapped applications to store 8-bit index values in the frame buffer.

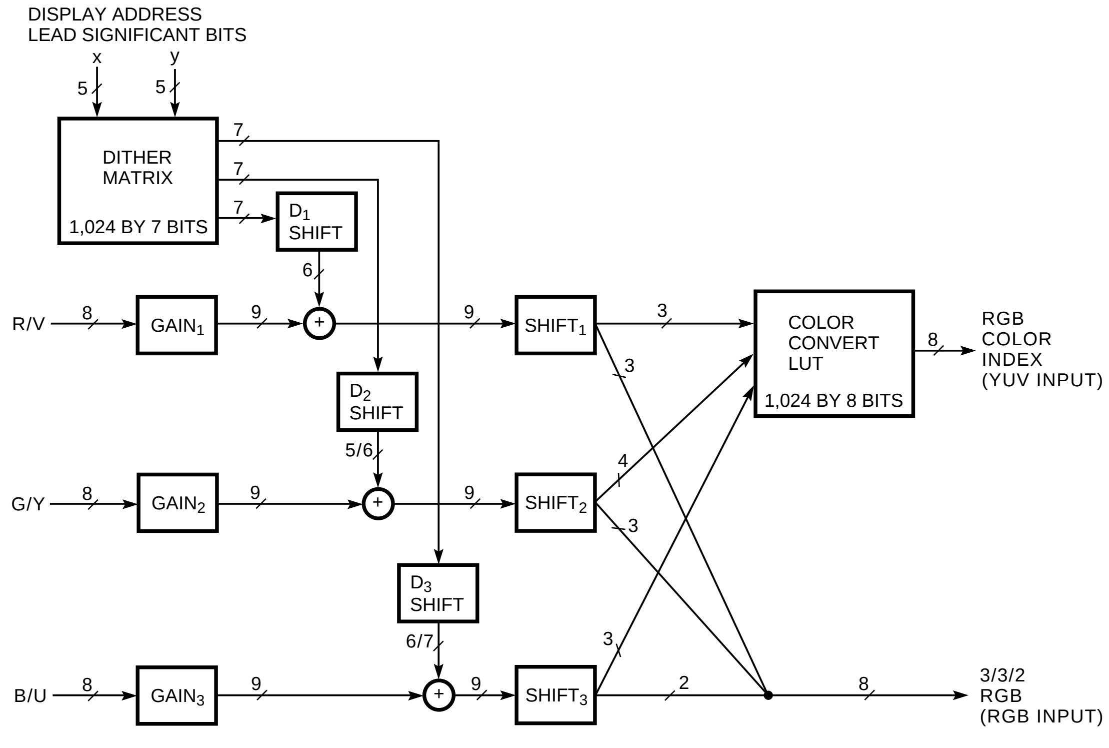

Figure 6
Dithering and YUV-to-RGB Conversion in the Dagger Chip

It is possible to extend this approach to use more bits of dithered YUV to produce more finely quantized RGB colors. The size of the required look-up ROM quickly gets out of hand, however. Dagger uses a 1K-by-8-bit ROM to convert 4/3/3 YUV into 256 RGB colors. Using 4/4/4 YUV would make the ROM five times larger (4K by 10 bits). To produce 4KRGB colors would require a ROM with 16K 12-bit entries.

YUV-to-RGB Color Conversion on TGA2 The TGA2 graphics chip performs dithering and color conversion in the reverse order, as compared to the Dagger chip. In TGA2, a YUV pixel is first converted into an RGB pixel at 8 bits per channel. This 24-bit RGB pixel is then either written to the frame buffer or dithered down to 8- or 12-bit RGB before being written to the frame buffer. Figure 7 shows the dithering system that is used in the TGA2 chip.

The key advantage of the TGA2 approach over the Dagger approach is that it allows deeper frame buffers to use higher-quality color conversion. If a 24-bit frame buffer is being used, TGA2 allows YUV to be converted to full 8/8/8 RGB. On the Dagger chip, YUV-to-RGB conversion produces only 256 different colors, regardless of the frame buffer depth. This is acceptable on Dagger, where 24-bit frame buffers are far from the design center. Also, the Dagger method uses fewer gates, which is an important consideration for the cost-constrained Dagger implementation.

Another advantage of this algorithm for TGA2 is that the set of colors used for video image display is the same one used by full-color synthetic graphics applications, such as a solid modeling package or a scientific visualization application. This allows a common color map to be used by both image applications and shaded graphics applications. Unlike the Dagger chip, TGA2 does not have an integrated RAMDAC and uses an external RAMDAC. Typical low-cost RAMDAC chips provide only one 256-entry color map, so it is important for TGA2 to allow image applications to share this color map with other applications.

Figure 8 illustrates how the TGA2 chip performs YUV-to-RGB color conversion. By the standard definition of the YUV format, the conversion to RGB consists of a 3-by-3 matrix multiplication operation in which three terms equal 1 and two terms equal 0.[2] The TGA2 chip performs this matrix multiplication using four LUTs to perform the remaining four multiplications, together with some adders. A final multiplexer is required to clamp the resulting values to the range 0 to 255.

The TGA2 color conversion algorithm has one disadvantage: the algorithm does not handle out-of-range YUV values as well as the technique used in the Dagger chip. In Dagger, each YUV triple that is out of range has an optimal or near-optimal RGB triple computed for it and placed in the table. With the TGA2 technique, the red, green, and blue components are computed separately. The individual color components are clamped to the range boundaries, but if a YUV triple results in an out-of-range value for green, this cannot affect the red or blue values. The result is some color distortion for oversaturated images. If such a result would be unsatisfactory, it is necessary to adjust the colors in software, e.g., by reducing the saturation or the intensity of the source image so that most YUV triples map to valid RGB colors.

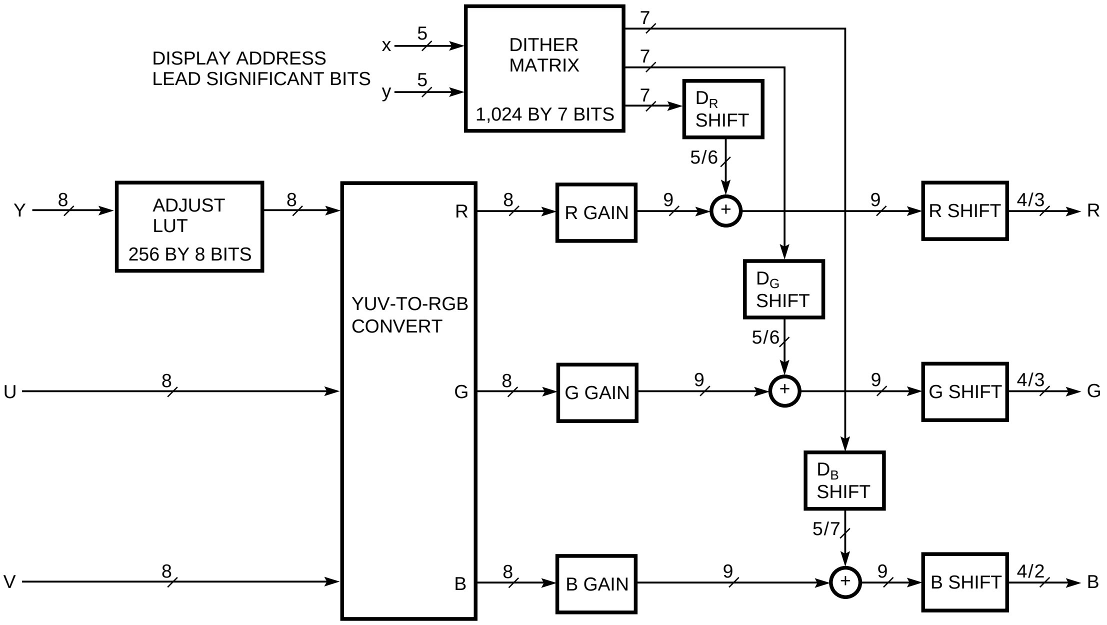

Figure 7
Dithering System in the TGA2 Chip

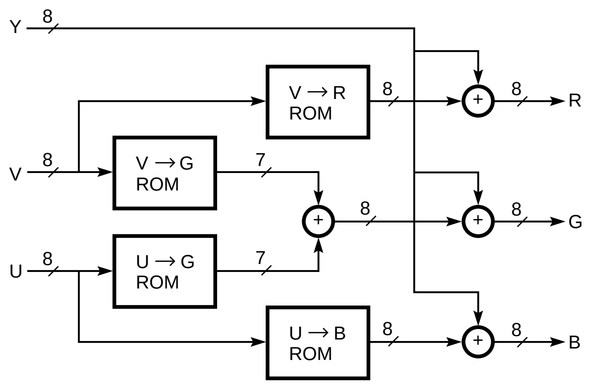

Figure 8
YUV-to-RGB Conversion in the TGA2 Chip

Table 2
Gates Used by the TGA2 Video Rendering Logic

Logic Block	Number of Cells	Number of Gates	Gates per Total Gates (Percent)
Pixel Formatting	778	584	4.2
Look-up Table	9,590	7,192	52.3
Filtering	2,265	1,699	12.4
Color Convert	3,486	2,614	19.0
Miscellaneous	2,210	1,658	12.1
Total	18,329	13,747	100.0

Implementation Cost and Performance

Both the Dagger and the TGA2 chips have the design goal of integrating as many as possible of the J300 design features into a single-chip graphics and video solution. Dagger and TGA2 include different features and implement some common features in different ways because each chip focuses on a different market. As mentioned earlier, Dagger is a PC graphics accelerator chip, and TGA2 is a workstation graphics accelerator chip.

Gate Cost

Table 2 shows the number of gates required to add the various imaging operations to the TGA2 chip. TGA2 is implemented in IBM's 5L standard cell technology. The video rendering logic represents less than 10 percent of the total TGA2 logic. The chip contains no additional gates for video scaling or dithering logic, since nearly all the gates needed to implement those functions are already required in TGA2 to implement Bresenham line drawing and dithering of 3-D shaded objects.

Table 2 clearly shows why the luminance adjust LUT was omitted from Dagger. On the TGA2 chip, the LUT requires more than half the total gates used for multimedia support.

Display Performance

The peak hardware performance for image operations on the TGA2 chip depends primarily on the internal clock rate, which is 60 megahertz (MHz). The TGA2 chip is fully pipelined, so that one pixel is processed on each clock cycle, regardless of the filtering, conversion, or dithering that is required. Reducing the image requires one clock cycle per source pixel. Enlarging the image requires one clock cycle per destination pixel. Actual hardware performance is never quite equal to peak rates, but TGA2 performance approaches peak rates. For example, TGA2's hardware performance limits support rendering a common intermediate format (CIF) image that is scaled up by a factor of three in both dimensions at over 30 fps.

Actual system performance depends on many factors besides hardware performance. Typically, multimedia images are stored and transmitted in compressed form, so that display performance depends on the speed of the decompression hardware or software. "Software-only Compression, Rendering, and Playback of Digital Video" contains tables that show the performance of a variety of AlphaGeneration systems with software-only rendering and with J300 rendering hardware that implements hardware algorithms similar to those in the TGA2 and Dagger chips.[5]

Table 3 shows the results of preliminary tests of TGA2 video display rates on AlphaStation 250 4/166 and AlphaStation 250 4/266 workstations, which use DECchip 21064 CPUs. The table shows performance in frames per second for displaying the standard Motion Picture Experts Group (MPEG) flower garden video clip, comparing performance to software algorithms that use the TGA graphics accelerator. Like TGA2, the TGA chip supports fast image transfers to the frame buffer; however, TGA does not provide any specific logic to accelerate video display.

The first two lines of Table 3 show performance for displaying images at their original size. Allowing TGA2 to convert decompressed YUV pixels to RGB improves performance by 34 to 45 percent, depending on CPU performance. This performance improvement drops to 18 to 25 when data transfer times are included. Possibly, this gap can be reduced by further coding to better overlap data transfer with MPEG decompression. Note that the TGA2 performance can include image filtering and a luminance adjust table lookup at no loss in performance.

The third line of Table 3 shows performance when the video clip is displayed at two times the size in both dimensions. The flower garden movie covers an area of 320 by 240 pixels, which is very small on a 1,280-by-1,024-pixel monitor. Therefore, it is highly desirable to display an enlarged image. In this case, TGA2

Table 3
Frames per Second for Displaying MPEG Flower Garden Video Clip

AlphaStation 250 4/166				AlphaStation 250 4/266		
TGA (fps)	TGA2 (fps)	Increase (Percent)		TGA (fps)	TGA2 (fps)	Increase (Percent)
24.7	35.8	45	Software decode rate	47.9	64.2	34
23.1	28.9	25	1x video playback rate	44.0	52.1	18
12.7	26.4	108	2x video playback rate	23.1	44.9	95

Source: Tom Morris, Technical Director, Light and Sound Engineering, Digital Equipment Corporation

displays the video clip at twice the speed of the software algorithm that uses the TGA graphics chip. The subjective difference is even greater, since TGA2 applies a smoothing filter to improve the quality of the resulting images. The software algorithm on the TGA chip performs no filtering because this would dramatically reduce chip performance.

The performance data in Table 3 are for displaying 8-bit images to the frame buffer. TGA2 is able to display 24-bit images at the same performance, up to the limit of its frame buffer bandwidth. For the examples in Table 3, TGA2 is able to produce either 8-bit, 12-bit, or 24-bit images at essentially the same performance. Software algorithms would experience a dramatic drop in performance, simply because they would have to process and transfer three times as much data. Therefore, the TGA2 chip allows significantly higher-quality images to be displayed without sacrificing performance.

Conclusions

This paper describes two graphics accelerator chips that integrate a set of image processing operations with traditional synthetic graphics operations. The image operations are carefully chosen to allow significantly higher performance with minimal extra logic; the operations that can be performed in software are left out. Both chips take advantage of the PCI bus to provide the bandwidth necessary for image data transfers.

The Dagger and TGA2 video rendering logic is based on the AccuVideo rendering pipeline as implemented in the J300 family of video and audio adapters.[3] The following restrictions were made to integrate this logic into these graphics chips:

1. Color preprocessing—Eliminate RAM for dynamic chrominance control. For the Dagger chip, also eliminate RAM for dynamic brightness/contrast control.
2. Filtering—Support just one sharpening and one smoothing filter (other than the identity filters) in the Dagger chip. For the TGA2 chip, support just two sharpening and two smoothing filters.
3. Color output—For the Dagger chip, allow only 256 output colors for YUV input [3/3/2 for RGB input]. For the TGA2 chip, support only RGB colors with a power-of-two number of values in each channel.

The quality of the resulting images is excellent. The AccuVideo 32-by-32 void-and-cluster dithering algorithm provides quality similar to error diffusion dithering algorithms.[8] Error diffusion is a technique in which the difference between the desired color and the displayed color at each pixel is used to control dithering decisions at adjacent pixels. Error-diffusion dithering requires considerably more logic than AccuVideo dithering and cannot be used when rendering synthetic graphics.

The high quality of the AccuVideo algorithm is especially important when dithering down to 8-bit pixels (3/3/2 RGB). Even in this extreme case, applying the AccuVideo dithering algorithm results in a slight graininess but few visible dithering artifacts. Applying AccuVideo dithering to 12-bit (4/4/4 RGB) pixels results in screen images that are almost indistinguishable from 24-bit (8/8/8 RGB) pixels.

We plan to continue evaluating new multimedia features for inclusion in our synthetic graphics chips. Areas we are investigating include more elaborate filtering and scaling operations, additional types of color conversion, and inexpensive ways to accelerate the compression/decompression process.

References

1. *PCI Multimedia Design Guide,* rev 1.0 (Portland, Oreg.: PCI Special Interest Group, March 29, 1994).

2. *Encoding Parameters of Digital Television for Studios,* CCIR Report 601-2 (Geneva: International Radio Consultative Committee [CCIR], 1990).

3. K. Correll and R. Ulichney, "The J300 Family of Video and Audio Adapters: Architecture and Hardware Design," *Digital Technical Journal,* vol. 7, no. 4 (1995, this issue): 20–33.

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

5. P. Bahl, P. Gauthier, and R. Ulichney, "Software-only Compression, Rendering, and Playback of Digital Video," *Digital Technical Journal,* vol. 7, no. 4 (1995, this issue): 52–75.

6. R. Ulichney, "Video Rendering," *Digital Technical Journal,* vol. 5, no. 2 (Spring 1993): 9–18.

7. R. Ulichney, "Method and Apparatus for Mapping a Digital Color Image from a First Color Space to a Second Color Space," U.S. Patent 5,233,684 (1993).

8. R. Ulichney, "The Void-and-Cluster Method for Generating Dither Arrays," *IS&T/SPIE Symposium on Electronic Imaging Science and Technology,* San Jose, Calif., vol. 1913 (February 1993): 332–343.

Biographies

Larry D. Seiler
Larry Seiler is a consultant engineer working in Digital's Graphics and Multimedia Group within the Worksystems Business Unit. During his 15 years at Digital, Larry has helped design a variety of graphics products. Most recently, he was the architect for the TGA2 graphics chip that is used in Digital's PowerStorm 3D30 and PowerStorm 4D20 graphics options. Prior to that he architected the SPX series of graphics options for VAX workstations. Larry holds a Ph.D. in computer science from the Massachusetts Institute of Technology, as well as B.S. and M.S. degrees from the California Institute of Technology.

Robert A. Ulichney
Robert Ulichney received a Ph.D. from the Massachusetts Institute of Technology in electrical engineering and computer science and a B.S. in physics and computer science from the University of Dayton, Ohio. He joined Digital in 1978. Bob is currently a senior consulting engineer with Digital's Cambridge Research Laboratory, where he leads the Video and Image Processing project. He has filed several patents for contributions to Digital products in the areas of hardware and software-only motion video, graphics controllers, and hard copy. Bob is the author of *Digital Halftoning* and serves as a referee for a number of technical societies, including IEEE, of which he is a senior member.