An Architecture for High-Performance 2-D Image Display

Stephen D. Jordan, Philip E. Jensen, Barthold B. A. Lichtenbelt

Hewlett-Packard, Graphics Hardware Lab and Graphics Software Lab
Ft. Collins, CO 80525

ABSTRACT

Image processing operations can be divided into two classes, those pre-processing operations that are market- and application-specific, and those widely-used operations that are useful in any application that requires the display of two-dimensional images. In the interest of achieving real-time rates for the broader class of 2-D image display operations, Hewlett-Packard has developed a hardware accelerator called VISUALIZE-IVX. It is capable of scaling, rotating, mirroring, translating and filtering 1k-by-1k output images at greater than 30 frames/sec while simultaneously enhancing image brightness and contrast. This paper describes the pipelined architecture used to achieve this performance on a desktop computer. The architecture makes use of a hybrid mapping scheme for geometric transformations. Also a unique memory device was designed that minimizes local image buffers while eliminating the need to resend pixels from main memory. A recently developed method of extending the filtering capabilities, that may be incorporated into future products, is also presented.

1. INTRODUCTION

The field of image processing encompasses a variety of image manipulation techniques that ranges from operations that are customized for specific applications to those that are more general-purpose. In general the former may be categorized as image pre-processing operations and the latter as image display operations.

The pre-processing tasks include image registration, pattern recognition, object segmentation, and frequency domain filtering. These application-specific techniques operate on large portions of the image simultaneously. They are therefore best accomplished with a software solution. The display operations include spatial filtering, scaling, rotation, translation, mirroring, and brightness and contrast manipulation. These operations are useful in almost any application. They use localized computations, i.e. they operate on pixel neighborhoods or single pixels. For these reasons it makes sense to accelerate them in hardware.

Hewlett-Packard has developed a 2-D image display accelerator called VISUALIZE-IVX which performs the image display operations at 40 Mpixels/sec. This paper discusses the system architecture and the design of the IVX that allowed us to achieve this performance cost-effectively.

The image display operations accelerated by the IVX can be classified as convolution (spatial filtering), interpolation (scaling, rotation, translation, mirroring), and window/level mapping (brightness/contrast control) operations. Full-speed performance (40 Mpixels/sec) has been achieved with *any or all* operations active. Furthermore, the order of operations can be changed without affecting performance.

This paper consists of 14 sections. Section 2 presents the overall system architecture. Section 3 lists the IVX design goals. Section 4 presents the IVX architecture, which in turn led to the development of the memory device described in Section 5. In Section 6 the unique geometric transformation technique is described. Then Sections 7 and 8 describe the pixel address generator and interpolator, respectively, that together implement the technique. The pipelined convolver circuit is described in Section 9. A method for supporting convolutions of arbitrary kernel sizes is described in Section 10. Section 11 contains a discussion of the window/level map circuit. Section 12 provides some chip-level details. The software used to expose the accelerator to the user is discussed in Section 13. The paper is summarized in Section 14.

2. SYSTEM ARCHITECTURE

We architected our 2-D image display system with the goal of achieving greater than 30 frames per second on 24-bit 1k-by-1k output images in a desktop workstation. This performance was to be achieved with an image display scheme that assumed the original image should be stored in system memory and passed through the display accelerator once for each frame to be rendered. This led to the choice of the HP PA-RISC SPU with the HCRX-24 graphics subsystem as the workstation platform. The IVX is a single printed circuit assembly that attaches to the HCRX as indicated in Figure 1.

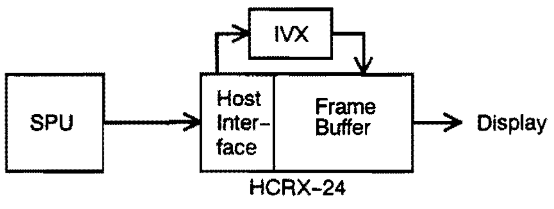

Figure 1. Workstation architecture

This system provides a high-bandwidth bus to the system memory, and sub-word parallelism in the CPU accelerates MPEG or other decompression techniques. The HCRX-24 uses an internal 45 MHz clock. By using this existing display subsystem we were able to leverage its existing production volumes and maintain a lower product cost.

3. IVX DESIGN GOALS

Our image display accelerator was designed with the following goals in mind. Along with each goal is listed the primary motivation for it.

- The accelerator should process more than 40 million pixels/sec independent of the number or order of internal operations active. This allows the system to achieve the desired objective of greater than 30 frames/sec on 1k-by-1k output images even with software and module interface overhead.

- The host CPU should be required to pass the source image to the IVX exactly once for each frame displayed. The intention here is to balance between minimizing the amount of CPU cycles spent moving data and minimizing the local memory required in the accelerator.

- Each IVX internal module should output processed pixels at a burst rate of 45 million pixels/sec. This is based on the first goal and the fact that the HCRX has an internal clock rate of 45 MHz.

- The order of image operations should be adjustable to allow customers in different markets to select the order most suitable to their application.

- Internal pixel precision should be maintained to ensure the integrity of 16-bit input data to the output image.

These design goals were met with the reconfigurable pipeline architecture described in Section 4, a unique local memory solution described in Section 5, and the individual module designs described in Sections 6-9 and 11.

4. IVX ARCHITECTURE

The IVX chip consists of three major components as shown in Figure 2. The convolver circuit uses a 3x3 programmable kernel to perform spatial filtering functions specified by the user. The address generator and interpolator pair can be programmed to support pan, zoom, rotation and mirroring about X or Y axes. The interpolator can operate in any of three modes: bi-cubic convolution, bi-linear, or nearest neighbor. Image brightness and contrast control is achieved with a RAM-based look-up table by the window/level map module.

One, two, or all three of the IVX functional blocks may be active at a time. In addition, except that the convolver can not follow the interpolator, the crossbar connecting the separate modules permits the order of operations to be adjusted arbitrarily.

Each block operates at a burst rate of 45 million pixels/second. To achieve the desired frame rate, the blocks are pipelined together: each module begins processing an image as soon as it has the minimum number of pixels to do so, and passes each output pixel to the next block in the chain as soon as it is ready.

The convolver and interpolator kernels both require pixels from multiple row to be available to compute the intensity of a single output pixel. This is accomplished with the line buffer memory described in the next section. This minimal memory is used to retain all pixels that are reused until they are no longer needed and thereby eliminates the need for the CPU to resend any pixels. The window/level map module does not need a local memory buffer since it performs a point process.

5. LINE BUFFERS

Pixels from three rows of the input image need to be simultaneously available to the convolver's math modules. The corollary is that each row of input pixels to the convolver will be used three different times. As depicted in Figure 3 this is achieved with a pair of memory modules called line buffers that store the reused pixels. Each line buffer holds a single row of input pixels and shifts a pixel into a convolver kernel register for each new output pixel computation. In this manner the convolver walks through each row of the input image from left to right. In a directly analogous manner the interpolator uses three line buffers to provide pixels from four rows of the input image to its 4x4 kernel. For both modules the line buffers are chained together such that one line buffer is fed from the incoming pixel stream and dumps it output to both the kernel registers and the following line buffer. With this chained arrangement the convolver or interpolator can sequence through the entire input image one row after another.

6. HYBRID MAPPING SCHEME

For the geometric transformations an efficient means of accomplishing the input to output mapping was needed. Neither of the traditional forward or inverse mapping algorithms lends itself to a low-cost high-speed hardware implementation. Therefore it was necessary for a new solution to be developed.

Forward mapping allows the input image to be traversed a single time in a straightforward manner, which permits an efficient stream-based design and requires only a minimal amount of memory to store a portion of the input image. However, a simple point-to-point mapping will cause holes and overlaps in the output image. The holes, of course, are unacceptable. The overlaps mean that an accumulator is necessary, which requires a large amount of memory and complicates the design. A region-based or four-corner mapping would eliminate the holes but would not eliminate the need for an accumulator.

Inverse mapping allows the output image to be traversed in a straightforward manner. This guarantees that all output pixels are computed (i.e. no holes), and eliminates the need for an accumulator. However, in addition to requiring an interpolation stage, it mandates a large memory buffer to store the entire input image. This method is used in texture mapping systems, where the interpolation must often be precomputed in the host, and additional local memory is typically used to store versions of the image interpolated at a number of different sample rates.

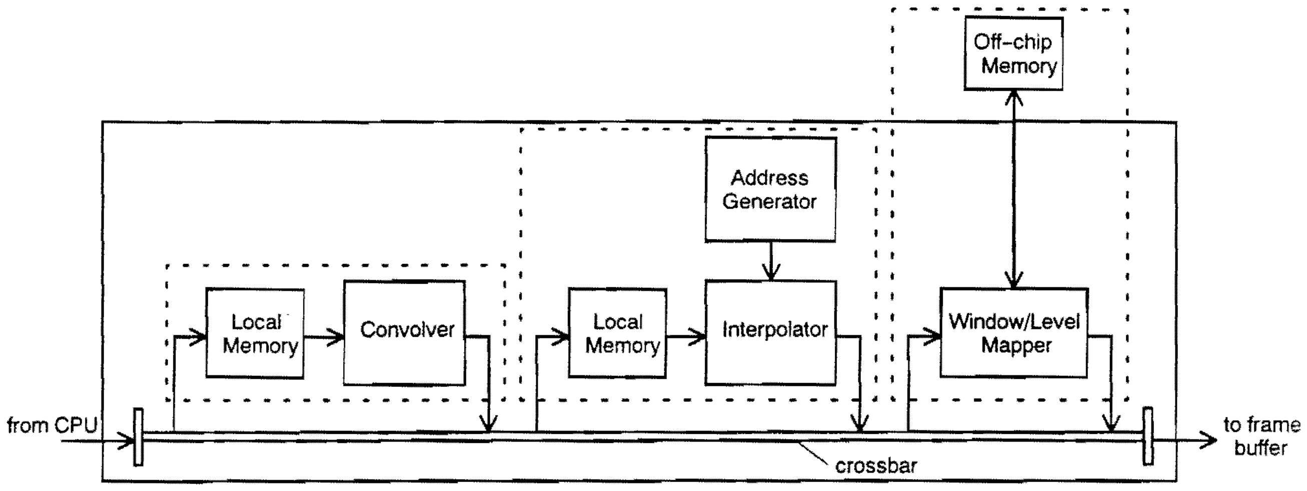

Figure 2. IVX Block Diagram

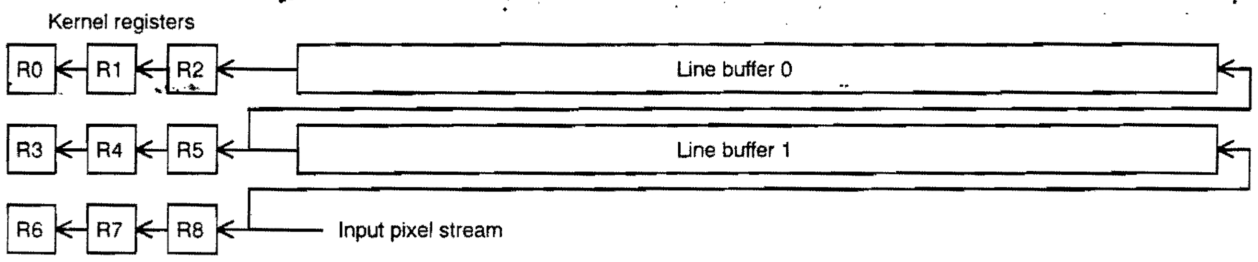

Figure 3. Convolver input data scheme

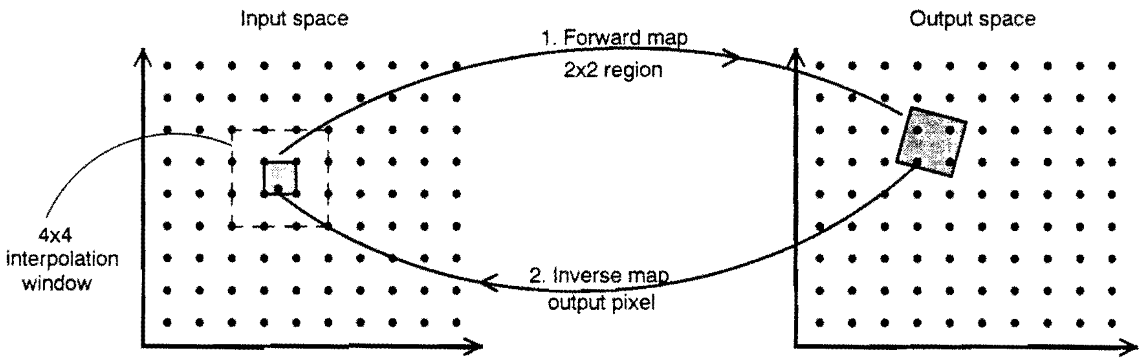

Figure 4. Hybrid mapping scheme

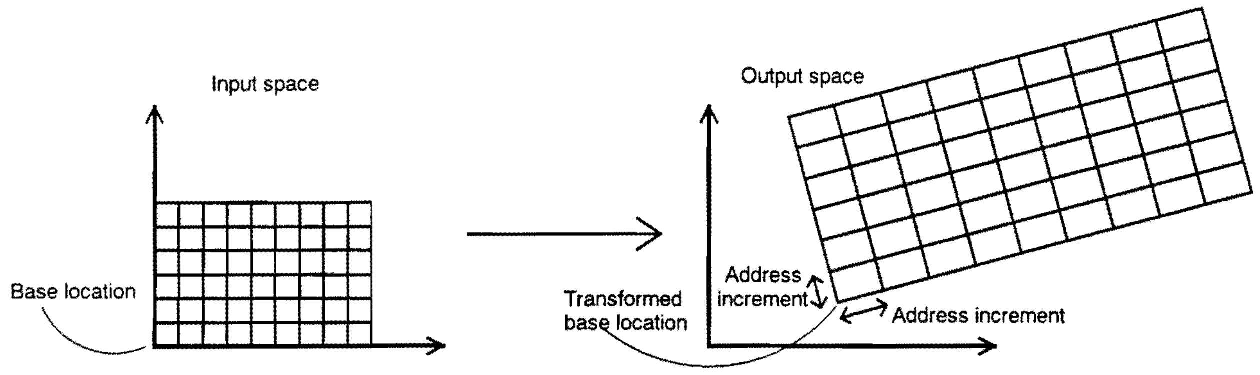

Figure 5. IVX address transformations

Our solution is to forward map a 2x2 input pixel subregion to determine its respective region in output space. Next the output space subregion is traversed in a manner that touches each output pixel exactly once. For each output pixel in a subregion an inverse transformation is performed to determine the exact relative location of the sample point in input space. Then an interpolation at the sample point is performed in input space using a 4x4 window to determine the respective output pixel intensity. This method is depicted in Figure 4 for an image that is translated, scaled by 1.8, and rotated clockwise 15°.

In this hybrid mapping scheme the input image is traversed a single time in a row sequential manner. This minimizes the local memory requirements and permits a pipelined architecture. The region-based output space traversal guarantees that all output pixels are computed, and means that an accumulator is not necessary, i.e. no memory is required for output pixels before the frame buffer. The implementation of this scheme in the IVX address generator and interpolator can perform affine transformations that preserve angles.

7. ADDRESS GENERATOR

In the IVX the forward mapping is accomplished by programming the IVX address generator module once per image with the transformed base address and the output space address increments. The transformed base address locates the point of the output image that corresponds to the first pixel in the input image. This output space point does not typically lie on an output pixel. The address increments specify the change in address in output space that corresponds to single pixel movement in the input image. These concepts are depicted in Figure 5. The example shown includes a translation, counter-clockwise rotation, and scaling where the scaling in X is greater than in Y.

Using the transformed address information the address generator determines all the output pixels within each output space subregion. (The subregion is the rectangular area bounded by a transformed input space 2x2 pixel group as shown in Figure 4.) The addresses of each pixel to be displayed is sent to the interpolator for the inverse mapping step. The order the address generator traverses the output space is constrained by two factors. Firstly, the interpolator only has pixels from four rows of the input data available at a time, and interpolation requires remaining between the center two rows. Secondly, the IVX frame buffer bus uses multiplexed address/data, so address cycles should be limited to minimize the image draw time. The address generator meets these constraints by drawing in a serpentine fashion within each transformed row region and just sending the step direction with each pixel. This is shown in Figure 6. The serpentine order of drawing by the address generator/interpolator pair is the reason the convolver can not follow them in the IVX pipeline, since the convolver requires its input data in scanline order. This is not an issue for the point processing window/level map circuit.

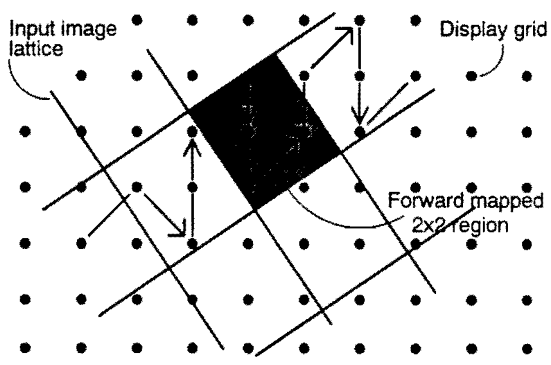

Figure 6. Address generator drawing path

Care must be taken in the computation of the location of subregion edges in order to ensure that no holes are created in the output image. In particular, the upper edge of one subregion must be situated at exactly the same location as the lower edge of the subregion just above it, even though these edges are computed at different times. This is readily accomplished by computing the starting address of an edge once and storing it for reuse the second time the edge is needed. Both times the edge is computed, from the starting address to the last subregion, identical computations are used by the address generator. This eliminates the possibility of different round-off errors each time the edge is used for clipping, and thereby ensures no holes will be created.

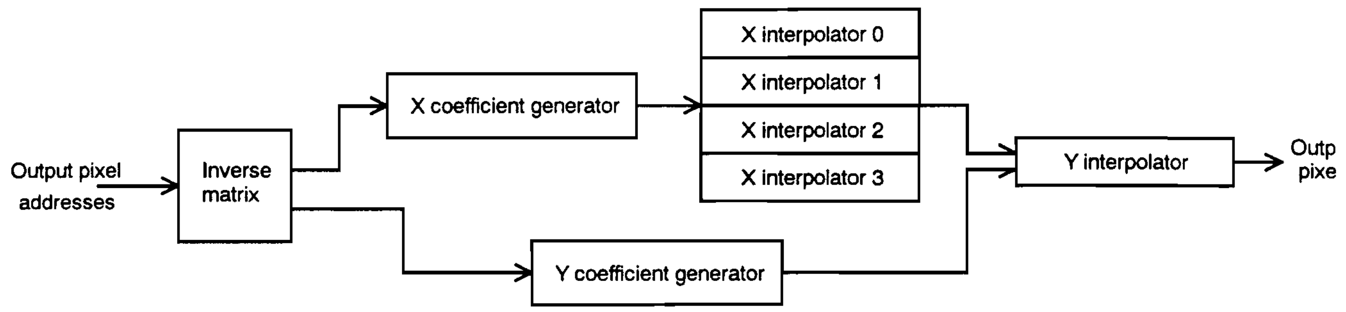

Figure 7. Interpolator block diagram

8. INTERPOLATOR

For each output pixel location calculated by the address generator, the interpolator module inverse maps to determine the location in input space and interpolates using the selected method (nearest-neighbor, bi-linear, or bi-cubic) to compute the output pixel intensity. For example, for bi-cubic a one-dimensional interpolation is performed on four pixels from each of four rows in the kernel. These interpolations occur in the input space X dimension. The results from the four rows are then used to interpolate in Y, and thereby compute the output pixel value. The 1-D interpolation function is described by Equation 1, where P_{int} is the interpolated pixel value, P_{i0}, P_{i1}, P_{i2} and P_{i3} are the input pixel values, and C_0, C_1, C_2 and C_3 are the coefficients or weights applied to the respective input pixels. The coefficient values are a cubic function of the distance from the sample point to the respective input pixel as described in Equation 2, where d_n is the distance from an input pixel n to the sample point in input space.

$$P_{int} = P_{i0} \times C_0 + P_{i1} \times C_1 + P_{i2} \times C_2 + P_{i3} \times C_3$$

Equation 1

$$\begin{aligned} C_n &= (A+2) \times |d_n|^3 - (A+3) \times |d_n|^2 + 1, && 0 \le |d_n| < 1; \\ C_n &= A \times |d_n|^3 - 5 \times A \times |d_n|^2 + 8 \times A \times |d_n| - 4 \times A, && 1 \le |d_n| < 2; \\ C_n &= 0 && \text{otherwise.} \end{aligned}$$

Equation 2

A block diagram of the interpolator is shown in Figure 7. The inverse matrix block is programmed once per image with the values of an inverse transformation matrix that maps the output pixel locations back to their relative locations in input space. The outputs of this block are the distances d_n of Equation 2 for the X and Y dimensions. The X and Y coefficient generators use a form of Equation 2 that takes advantage of the relationship between the four d_n values to use less multipliers and adders.[1] This implementation has nine multipliers and seven adders each for X and Y. Each X/Y interpolator is a straightforward implementation of Equation 1 that produces a 16-bit pixel result. The X interpolators receive the input pixels using a line buffer scheme analogous to that shown for the convolver in Figure 3.

The interpolator submodules are chained together as a single pipelined unit. All told there are 82 multipliers and 64 adders in the three-channel (red, green, blue) interpolator pipeline. Once the pipeline is loaded, new data can be shifted in and a new pixel result can be calculated in every clock cylce. With the 45 MHz clock, this gives a burst pixel interpolation rate of 45 million pixels/second. Bi-linear and nearest-neighbor interpolation modes are handled by switching in appropriate values for the coefficients, so the interpolator runs at the same speed in these modes.

The address generator and interpolator work together to perform geometric transformations. X and Y scaling by integer and non-integer factors up to 32 are supported. Any ratio of scaling in X to scaling in Y from 1/2 to 2 is allowed. The angle of rotation may be any multiple of 1/10th degree from 0 to 359.9. Images can be translated to any position resolved to less than 1/1000th of the pixel-to-pixel distance. The modules operate together at a burst rate of 45 million pixels/second.

9. CONVOLVER

The convolver circuit is a spatial filter using a 3x3 kernel with programmable coefficients. For a given pixel in the input image, the corresponding output pixel value is a function of the input pixel and its eight adjacent pixels. Each of these nine input pixel values is multiplied by the respective coefficient in the convolution mask.

The coefficients of the mask can be programmed once for any single image. By selecting the appropriate values for the mask coefficients the convolver can be used to perform low-pass filtering (blurring), high-pass filtering (sharpening), edge enhancement, or other functions determined by the user.

The convolution is performed using the math modules shown in Figure 8. The modules are pipelined so that an output pixel is calculated for every clock period, and the convolver can therefore generate 45 million output pixels/second once the pipe is filled.

Since each output pixel is dependent on its eight neighbors, special means should be provided for calculating output pixels at the edge of the image. The IVX convolver includes hardware support for expanding the input image size by one at each edge using a constant pixel value or by copying the pixels at the edge. Alternatively, custom adjustments may be made to the edges of the image before passing it to the IVX, and the convolver can allow the image size to shrink in each dimension by not locally adding data to the edges.

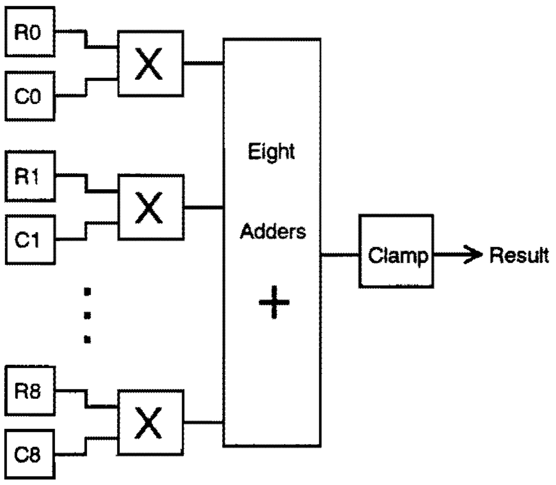

Figure 8. Convolver math data flow

10. CONVOLVER ENHANCEMENTS

Realizing that support for convolutions of larger kernel sizes would be useful, some work has been done since completing the IVX for potentially enhancing convolution in future products.

Simply increasing the number of multipliers and adders to the convolver to support larger kernels quickly becomes expensive in terms of chip real estate. However, by increasing to a core 5x5 convolver circuit, and using it in multiple passes in conjunction with a local memory buffer, performing convolutions of arbitrarily large kernel sizes is possible. For example, 7x7 convolutions can be performed in two passes by computing 25 of the 49 pixel-coefficient products in the first pass, storing the intermediate sum for each pixel in the memory buffer, and performing the remaining 24 products and final summing in the second pass. A 7x7 convolution can thereby be performed at fully half the speed of a 3x3 or 5x5 convolution.

This idea can be extended further. By using more passes through the input image even larger kernel sizes can be supported using the same 5x5 core. For example, an 11x11 convolution can be performed by breaking the kernel into six sections and computing up to 24 pixel-coefficient products in each pass. Larger kernel sizes take correspondingly more time for the convolution process, but the performance advantage of the hardware accelerator over a CPU software-based spatial convolution remains constant.

11. WINDOW/LEVEL MAPPER

Window and level controls are essentially pixel contrast and brightness controls. As such they are straightforward but powerful means of image transformation. By implementing with a standard mapping table any arbitrary mapping function can be used. Since most computer displays use an 8-bit DAC per channel, this is a useful enhancement to make to the 16-bit image data before sending it to the display in order to make use of the reduced range effectively. Alternatively a mapping function could be loaded that converts single-channel input to red/green/blue output.

The IVX includes a full 16-bit input to 32-bit output mapping table which is stored in high-speed static RAM. The window/level mapper module in the IVX chip controls accesses to this RAM, piping up the table look-up reads to achieve a 45 million pixel/second throughput rate. The module supports accessing the off-chip RAM as a single 8-bit LUT for fast loading, a single 16-bit LUT for maximum control, or sixteen 12-bit LUTs. The third mode allows consecutive images to be rendered more quickly by pre-loading the sixteen tables and just switching between them for every frame.

12. CHIP-LEVEL STATISTICS

The IVX chip was implemented using Hewlett-Packard's proprietary 0.45 μm CMOS process and includes 1.7 million transistors. All math modules are 24 bits or wider to maintain internal precision of 16 bits/channel. Several statistics for the chip are listed in Table 1. It is interesting to note that in this day of 100+ MHz clock speeds that we have been able to achieve industry-leading image processing performance without such a high chip frequency.

Table 1

Parameter	Value
Die size	14.2mm × 14.2mm
Pins used	348
Power consumption	<4 watts
Multiplier count	96
Adder count	85
FET count	1.7 million
Interpolator FET count	1.1 million
Address Generator FET count	0.18 million
Convolver FET count	0.16 million
Internal memory	44 kbits
Clock speed	45 MHz

13. VISUALIZE-IVL SOFTWARE

A hardware system is unusable without the accompanying software that drives it. The IVX hardware is accessible to the user via an imaging library called VISUALIZE-IVL which uses an API based on the imaging portions of OpenGL® and the imaging extensions thereof. This has been shown to be an effective strategy for an imaging API.[2] If the IVX accelerator is not present in the system, IVL will automatically fall back to a software solution, with a performace penalty. In either case IVL is optimized for memory to display transfers of image data.[3]

While the OpenGL® API is not often thought of as an API for imaging, it was designed to expose the capabilities of modern frame buffer hardware. The emphasis in this API is on 3D graphics, but it also includes a fairly rich set of capabilities for 2D imaging. The core capabilities of the OpenGL® API can be extended using extensions for imaging proposed by Silicon Graphics, Inc. and others.

IVL is a stand-alone library that implements the imaging portions of the OpenGL® API and some of the OpenGL® imaging extensions. IVL is a low-level application programming interface. By this, we mean that IVL is intended to provide access to the frame buffer with the highest possible performance and the lowest possible overhead. IVL does not include elaborate image processing algorithms, nor does it support high-level abstractions for image formats. The API focuses on providing a highly efficient path for transferring pixels to and from the frame buffer. Unlike X, IVL provides applications direct access to the frame buffer hardware without the need to go through any intermediate software layers or protocols.

Even though IVL exposes the capabilities of the IVX hardware, it is not a hardware-specific API. It provides some degree of abstraction from the hardware in order that it might be implemented on a variety of hardware platforms, even on devices that have no specific acceleration hardware for pixel processing operations. This level of abstraction allows software written on top of IVL to be ported easily to any system that supports IVL.

With its similarity to the OpenGL® API, software written using IVL can be easily ported to an OpenGL® environment. IVL provides a small, well-defined set of capabilities for pixel processing. The IVL entry points are identical in syntax and semantics to their counterparts in OpenGL®. IVL provides API calls that are similar to function calls in the following OpenGL® extensions:

- EXT_convolution
- EXT_texture
- EXT_visual_info
- EXT_color_table
- HP_image_transform
- HP_convolution_border_modes

Extensions that begin with "EXT" have public support from two or more companies. This typically means that at least two of the companies licensing OpenGL® technology are shipping an OpenGL® product that supports the extension, or that they are planning to do so. The HP_image_transform extension addresses the lack of image transformation facilities (zoom, rotate, resample. i.e. the 2-D affine transformation) in the OpenGL® pixel processing pipeline. The HP_convolution_border_ modes extension adds additional border-handling methods to the EXT_convolution extension.

14. SUMMARY

An image processing architecture has been described that can perform the most widely used image transformations in real time. This has been achieved while minimizing the load on the the system CPU. The accelerator performance goals were met by chaining together dedicated high-speed modules that each perform one or more of the desired transformations. Except at the very beginning and end of the image, all modules are simultaneously busy, operating in parallel albeit on different portions of the image. High throughput rates are thereby sustained even when multiple operations are performed on the same image, and independent of the order of operations. All operations maintain 16 bits/channel internal precisions by using wider math modules throughout the pipeline.

In order to achieve these results a unique image transformation algorithm was developed for the geometric transformations. This algorithm uses a hybrid of forward and backward mapping to perform affine transformations efficiently. Also, a unique memory scheme was described that permits the image to be sent down only once from the host, but minimizes local memory requirements.

Including image setup, driver and API overhead, the IVX has been measured performing *simultaneous* filtering, scaling, translation, and windowing/leveling on a 256x256 image scaled to 1kx1k at 43 frames/second.

15. ACKNOWLEDGMENTS

The authors would like to acknowledge the contribution of the entire hardware and software teams in the successful development of the VISUALIZE-IVX and VISUALIZE-IVL products.

16. REFERENCES

1. S. D. Jordan, "High-performance image processing on the desktop", *Medical Imaging 1996: Image Display,* Yongmin Kim Ed., SPIE Proc. Vol. 2707, May 1996.

2. R. J. Rost, "Using OpenGL for Imaging", *Medical Imaging 1996: Image Display,* Yongmin Kim Ed., SPIE Proc. Vol. 2707, May 1996.

3. D. A. Desormeaux, "The secrets of high-performance image display", *Medical Imaging 1996: Image Display,* Yongmin Kim Ed., SPIE Proc. Vol. 2707, May 1996.