Finding Lines and Building Pyramids with Splash 2

A. Lynn Abbott, Peter M. Athanas, Luna Chen, and Robert L. Elliott

The Bradley Department of Electrical Engineering
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061-0111

Abstract

This paper describes the design and implementation of two image-processing algorithms using the SPLASH 2 *custom computing platform.* SPLASH 2 *is a reconfigurable system that can be tailored to perform a wide variety of tasks. The particular tasks discussed here are the Hough transform, a well-known technique for detecting lines in an image, and pyramid generation, the process of transforming a single image into a set of filtered images with successively lower spatial resolution. This paper describes how these computationally intensive processes have been mapped onto* SPLASH 2 *hardware. Both processes have been designed to operate at high speed. In particular, the generation of both Gaussian (low-pass) and Laplacian (band-pass) pyramids can occur concurrently in real time using images from a video camera, assuming the standard frame rate of 30 images per second. Results are presented to illustrate the efficacy of reconfigurable FPGA-based machines to image processing applications.*

1 Introduction

General-purpose workstations can provide acceptable performance for many image-processing tasks when high speed is not required. However, for such applications as robotic control, visual tracking, and autonomous navigation, fast image processing is essential. Unfortunately, fast computation rates are difficult to achieve because of the large amount of data associated with images. General-purpose machines are overwhelmed not only by the numbers of computations required, but also by the I/O requirements of real-time image processing, which exceed 7 Megabytes per second for typical monochrome video cameras.

When high performance is needed, the traditional approach has been to develop application-specific hardware. The drawbacks of this approach are lengthy design cycles and costly redesigns when new algorithms are developed (or when mistakes are found in the original design). An attractive alternative is to utilize reconfigurable FPGA-based platforms that can be tailored to different applications without sacrificing performance.

This paper examines the computational benefits of the reconfigurable approach by presenting two case studies using the SPLASH 2 attached processor. A brief overview of SPLASH 2 is given in Section 2. The two tasks that are presented are well known in the image-processing community, and are substantially different in nature. The first of these, described in Section 3, is a 23-FPGA design to perform the Hough transform and related preprocessing functions. Section 4 describes two 10-FPGA designs for generating real-time (and near real-time) image pyramids at the standard video rate of 30 images per second. Section 5 is a comparison and summary of these designs.

2 The Splash 2 Architecture

SPLASH 2 is a second-generation reconfigurable attached processor designed by the Supercomputing Research Center in Bowie, Maryland. This section presents a brief overview of the system; refer to [Arno92] for more details.

A SPLASH 2 system consists of a Sun SPARC-2 host, an interface board, and one or more array boards. Up to 15 SPLASH 2 array boards are supported within a single system, and each contains 17 Xilinx XC4010 FPGAs arranged in a linear array and are fully connected through a 16x16 crossbar. (Refer to Figure 1.) The 17 FPGAs are designated X0 - X16, and each has a local 256Kx16 RAM. The principle data paths between FPGAs are 36 bits in width, and can therefore accommodate up to four 8-bit picture elements (pixels) simultaneously along with a 4-bit control word.

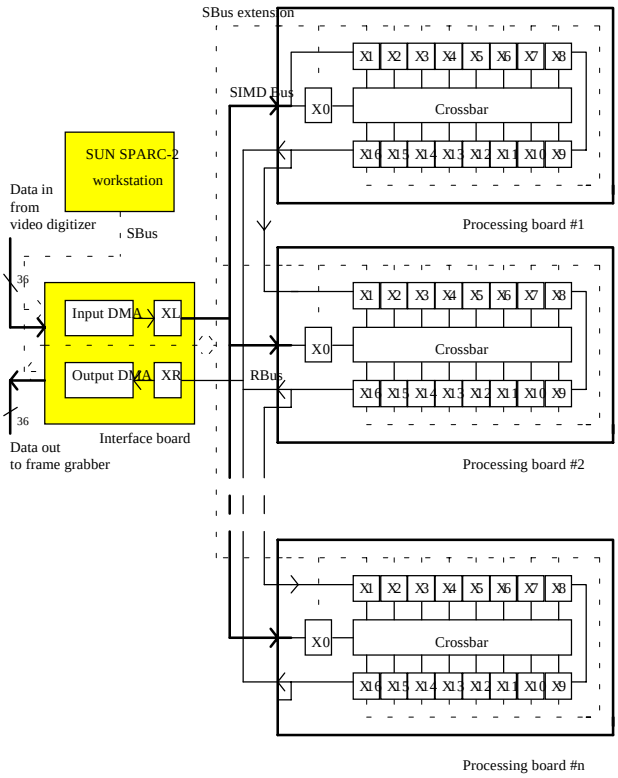

Figure 1. The SPLASH 2 architecture. Up to 15 reconfigurable array boards can be placed in a system. Each board contains 17 Xilinx FPGAs that are connected as a linear array and through a crossbar.

Applications are "programmed" for SPLASH 2 by creating a VHDL model which is simulated, refined, and synthesized to a gate list. An alternative method is to use schematic capture and supporting software such as XBLOX [Xili91]. The gate list is finally mapped onto the FPGA architecture using the Xilinx or the Synopsis synthesis tools. This design cycle is described more thoroughly in [Arno93].

3 Line Detection using the Hough Transform

3.1 The Hough Transform

The Hough transform is a procedure for detecting straight lines in an image [Houg62, Duda72]. Line detection is a very useful low-level operation, particularly for image analysis of scenes that may contain man-made objects such as buildings, roads, and manufactured goods. Extensions of the method may be used to detect curves or other objects of known shape.

The essence of the method is to transform edge points in an image to curves in *parameter space*. An accumulator array is maintained to contain these curves. After all edge points have been processed, peaks in the accumulator array (also called the "Hough array") indicate the parameters of the lines in the original image.

To see this, consider the following equation for a line: $d = x\cos\theta + y\sin\theta$. The variables x and y correspond to horizontal and vertical image dimensions, respectively. As illustrated in Figure 2a, the value d represents the perpendicular distance of the line from the origin, and θ is the angle of the perpendicular with the x-axis. (The standard slope-intercept equation is not used because of problems with vertical lines.) The problem of line detection is to recover the two parameters d and θ that pass through a significant number of edge points in the image. This is done by mapping each edge point (x_i, y_i) in the image to a sinusoid in d-θ parameter space, as illustrated in Figure 2b. For collinear edge points, all of the corresponding sinusoids will intersect at the single point (d_j, θ_j) which corresponds to the line through the edge points.

The standard Hough approach implements the parameter space as an accumulator array, where each quantized d-θ cell in the array corresponds to a single line in the image. Each cell in the array is initialized to 0. For each edge point, the corresponding sinusoid is traced within the Hough array, incrementing each cell that is visited. In effect, each edge point "votes" for all lines that pass through it. After all edge points have been processed in this manner, the array is examined. Any large values are good candidates for lines that may be present in the image.

3.2 Implementation

When detecting whether lines are present in a real visual data stream, only boundaries of objects within the visual field are of interest. In other words, if one wanted to characterize a gear or a wrench with a set of lines, the colors and textures of these objects are not relevant. In order to accentuate object boundaries, the image is first processed with a 3×3 Sobel edge detector[1]. The resulting high-pass data are

[1]This function, along with the other pre-processing operations, are also implemented on SPLASH on a separate processor board, and also operates in real-time on the input data stream.

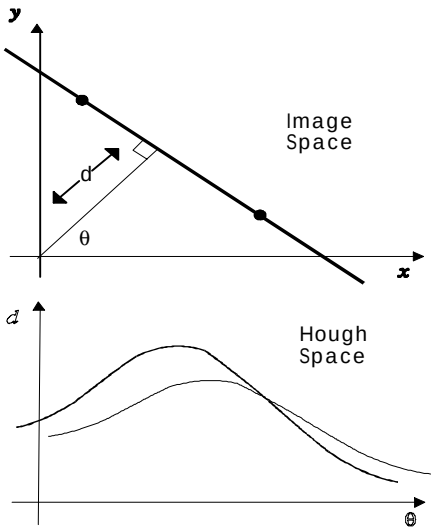

Figure 2. The Hough transformation to parameter space. Edge points (x_i, y_i) in the image (a) map to sinusoids in the d-θ *parameter space (b). In this example, the two sinusoids intersect at the values* d *and* θ *which determine the line that passes through (x_1, y_1) and (x_2, y_2).*

filtered and thresholded to eliminate outlyers. The digitized raster-scan video signal consists of 512×512 8-bit pixels, which is clocked into SPLASH 2 at a rate of 10 million pixels per second.

This filtered binary data stream tends to be relatively sparse in '1's, which are present only around object boundaries. Figure 3 provides a high-level overview of the complete tranformation process on the laboratory platform. The focus of this section is on the construction of parameter space (*BOARD 2* in Figure 3).

The operations that are to be realized in the SPLASH 2 platform are as follows: 1) extraction of *x* and *y* values for edge points; 2) derivation of values for θ; 3) computation of $x\cos\theta$; 4) computation of $y\sin\theta$; 5) calculation of *d* by adding the partial sums; 6) forming an address into the accumulator array for every *d*-θ pair; and 7) incrementing the value stored at that address. These operations have been mapped to SPLASH 2 devices X0-X16 as shown in Figure 4. The components of this are described below.

There are fourteen Hough processors on a SPLASH 2 processor board. In addition, one processing element (PE) serves to queue the frame data (since throughput is data dependent), and another PE serves as a task dispatcher. The task dispatcher examines each pixel value. If the pixel is an edge point, the pixel's column

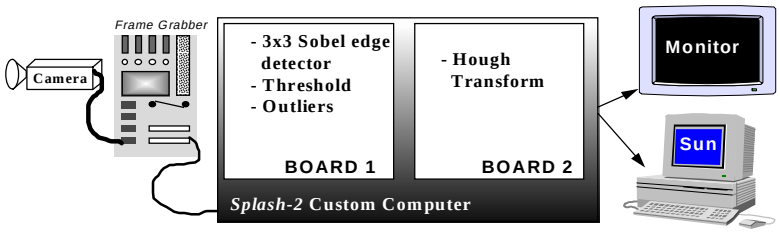

Figure 3. The laboratory system use for the Hough transform implementation.

and row within the image are passed to an available Hough processing element. The row and column serve as *x* and *y* values, respectively. This PE also serves to broadcast the appropriate *sine* and *cosine* factors to the slave PEs. All of the PEs are in lock-step.

Within each Hough PE, the generated Hough array is stored in a static RAM (attached to each PE), and is quantized to 512 values of θ (corresponding to the range [-90°, 90°) in steps of 180/512 degrees) and 512 discrete values of *d* (covering the range [-722, 721)). Each Hough processor performs the computations $x\cos\theta$ and $y\sin\theta$ in a four-step process. Each PE accepts the two 9-bit quantities *x* and *y* (indicating the address of the edge point in image space), uses a 9-bit counter to increment from 0 to 511 for each quantized value of θ, accepts broadcast values for $\cos\theta$ from the task dispatcher, and performs a 13×13-bit signed multiplication to compute $x\cos\theta$. Likewise, values for $\sin\theta$, are accepted from the task dispatcher, and another 13×13-bit multiplication is performed to compute $y\sin\theta$. An adder is used to generate $d = x\cos\theta + y\sin\theta$. An address into the Hough array is generated by concatenating the 9-bit quantity θ and a scaled 9-bit quantity *d*. This 18-bit value is used as an index into the attached RAM. A 4-clock read-modify-write operation is used to increment the indexed RAM location. As mentioned earlier, all of these operations are combined into a 4-clock pipeline.

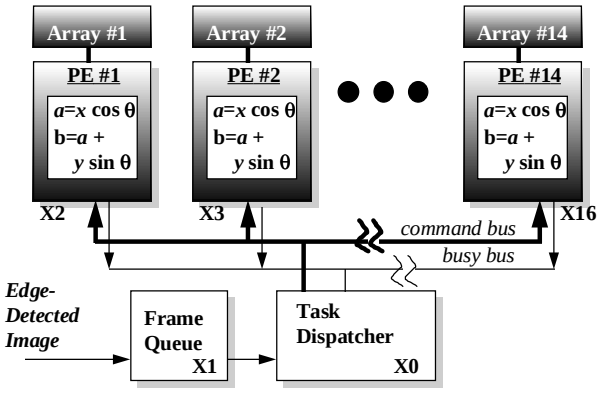

Figure 4. Architecture for the Hough transform.

3.3. Discussion

The amount of processing required by the Hough transform is data dependent. For each edge point in the image, a constant number of calculations are needed for tracing through the accumulator array. Post-processing is needed to combine separate Hough arrays from each PE, and to detect peaks in the composite array. Considering only arithmetic operations (disregarding internal data movement), and neglecting operations associated with input edge detection, this task performs approximately 310 million operations per second.[2] This application is easily extendible to multiple SPLASH processor boards for higher throughput.

A sample image processed by the Hough transform is provided in Figure 8.

4 Pyramid Generation

4.1 Image Pyramids

Multiresolution image processing techniques have emerged over the past decade as important components of advanced image analysis systems. Such computer vision tasks as stereo matching, motion analysis and object recognition routinely utilize image information at different levels of detail. This is done for reasons of efficiency and simplicity. For example, the photograph of a building may consist of large-scale objects such as walls, small-scale objects such as doorknobs and bricks, and medium-scale objects of intermediate size. These different entities can best be detected by using techniques that operate at the appropriate level of image resolution.

A standard data structure which contains image data at different levels of resolution is the *image pyramid*. An image pyramid is constructed by recursively filtering and subsampling an input image, creating a hierarchy of images of decreasing size and spatial resolution. Although a number of pyramid types have been developed, the most popular are the Gaussian and Laplacian pyramids [Burt82] which are illustrated in Figure 9. Each image in a Gaussian pyramid is formed by smoothing and subsampling the image at the previous level. Laplacian pyramids contain band-pass versions of the images, so that each level represents intensity edges at a particular resolution..

[2]Formal definitions of MIP ratings (or other benchmarks) for custom computing machines remain undefined.

After a pyramid has been constructed for a given image, analysis can begin at the lower-resolution portion of the pyramid to guide processing at higher-resolution levels. For some tasks (such as surveillance and road following) this approach can greatly reduce the overall amount of processing required.

4.2 Gaussian Pyramid Generation

The standard technique for generating a Gaussian pyramid is described in [Burt82]. The original image g_0 is processed to generate the lower-resolution image g_1, which is in turn processed to generate the lower-resolution image g_2, etc. The sequence of images g_0, g_1, . . ., g_{N-1} is called the Gaussian pyramid. Each image g_k is at half the resolution (in both dimensions) as g_{k-1}, so that the number of pixels at each level is one quarter of the number at the previous level.

This is illustrated in Figure 5. Each pixel value in g_k is obtained by a weighted sum of pixels from g_{k-1}, computed over a 5x5 neighborhood as follows:

$$g_k(i,j) = \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m,n)\, g_{k-1}(2i+m, 2j+n).$$

For simplicity, this process is referred to as REDUCE when computed for all values *(i, j)* at level *k*:

$$g_k = \text{REDUCE}\,(g_{k-1})$$

To simplify computational requirements, the 5x5 filter *w* is typically made separable: $w(m,n) = w_x(n)\, w_y(m)$. In addition, it is usually normalized, symmetric, and unimodal so that it resembles the Gaussian function. The weighting patterns that have been chosen for SPLASH 2 are [1/16 1/4 3/8 1/4 1/16] for w_x and the transpose of this for w_y. Because *w* is separable, an image can be filtered by the one-dimensional (1D) filters w_x and w_y in sequence. The two 1D filters are considerably simpler to implement in hardware than the single 2D filter that they replace. Powers of 2 have been chosen for the denominators so that filtering can be accomplished using only shift-and-add circuitry.

4.3 Implementation for Gaussian Pyramids

A block diagram of the SPLASH 2 implementation for real-time Gaussian pyramid generation is given in Figure 6. The FPGA designated as X0 simply receives image pixels and passes them to FPGA X1 through the crossbar. Four 8-bit pixels are presented within each

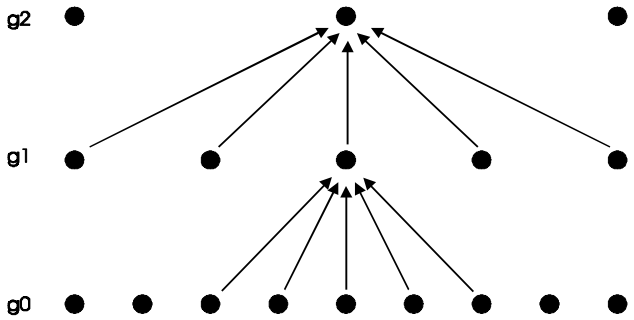

Figure 5. One dimensional representation of Gaussian pyramid generation. Each row of dots represents an image within one level of the pyramid. Each dot represents one pixel, which is computed as the weighted average of 5 pixel values from the level below it. Notice that the pixel spacing doubles from one level to the next, while the same weighting pattern is used to generate all new levels. (Adapted from [Burt82].)

36-bit word, and the input data is updated every four SPLASH 2 clock periods. The processing steps, in sequence, are horizontal convolution by w_x, (X1), vertical convolution by w_y (X2 and X3), storage of g_1 (X4), generation and storage of g_2 through g_4 (X5-X8), and output selection (X9). Image resolutions are reduced from 512x512 (for g_0) to 32x32 (g_4). X1 performs the convolution by w_x, resamples to eliminate half of the pixels per image row, and passes the result to X2. Some buffering is required because each computation involves the weighted sum of *five* pixels, whereas only *four* are received at a time.

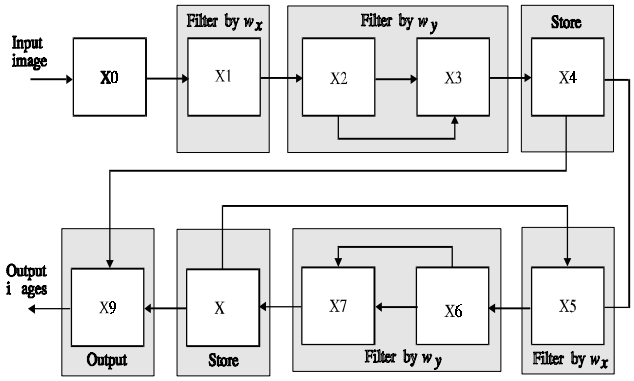

Figure 6. SPLASH 2 implementation of a 5-level Gaussian pyramid generator. This design can generate the upper 4 levels at video frame rates of 30 images/second. Xilinx chips X5-X8 duplicate the circuitry in X1-X4.

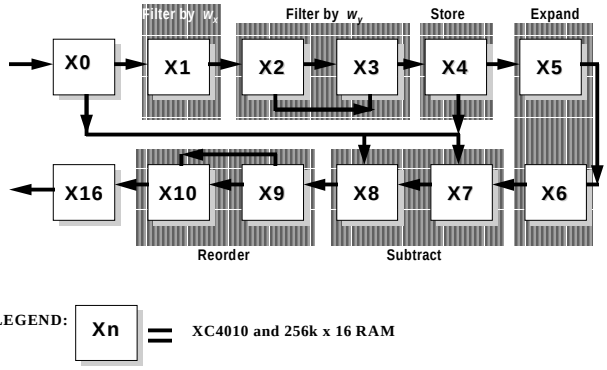

Figure 7. SPLASH 2 implementation for simultaneous Gaussian and Laplacian pyramids.

Devices X2 and X3 work together to perform the convolution by w_y. Each computation requires five image pixels from five successive rows. This requires that four previous rows must be stored and accessed simultaneously. To accomplish this, X2 and X3 redundantly store image rows in their respective RAMs. The crossbar is used to pass the untouched pixel stream. X2 computes three of the five partial sums, and passes this result to X3 directly. X3 adds this to the remaining two partial sums. This result is passed to X4, which resamples the image in the vertical dimension and stores the complete image g_1 in memory.

Xilinx devices X5-X8 generate the remaining three levels of the pyramid. The design is such that these devices duplicate the circuitry contained in X1-X4. Because of the reduction in image sizes, each successive level of the pyramid takes one forth of the time required at the previous level. Working at the same clock rate, X5-X8 are able to generate g_2 through g_4 (by recirculating from X8 to X5) in less than one third the time required to generate g_1. X9 multiplexes the results from X4 and X8 to deliver g_1 through g_4 as output, in sequence, simultaneously while the next image is entering the pipeline.

4.4 Laplacian Pyramid Generation

A Laplacian pyramid may be described as a sequence of "difference images," where each image is the difference between two successive levels of the Gaussian pyramid [Burt82]. To accomplish this, the resolution of the image at level *k+1* must be increased to the resolution at level *k* before subtraction can occur. This operation is known as EXPAND, and is accomplished by up-sampling the image produced at

level *k+1* (by inserting zeros between each sample), and interpolating with a second *reconstruction* filter. Like the REDUCE operation, the reconstruction filter is implemented using a 5×5 separable window filter.

If the Laplacian image at level *k* in the pyramid is represented by h_k, then operation to be performed may be written as $h_k = g_k - \text{EXPAND}(g_{k+1})$. Since image g_N does not exist, it is customary to define $h_{N-1} = g_{N-1}$.

4.5 Implementation for Simultaneous Gaussian and Laplacian Pyramids

Additional circuitry is needed when the Laplacian pyramid is to be generated along with the Gaussian pyramid. The design discussed here requires six more FPGAs, placing the total above those available on one SPLASH 2 board. It was decided to reduce the computational speed to half of video frame rate by devoting X5-X8 to Laplacian pyramid generation (rather than to the Gaussian pyramid as in the earlier section).

This is illustrated in Figure 7. Devices X1-X4 operate as before, except that X4 can now feed back through the crossbar to X1 to permit recirculation of Gaussian pyramid levels. By adopting this approach, only alternate video frames will be processed. X5 and X6 are now devoted to the EXPAND operation. X7 and X8 accept data from X4 and X6 and perform pixel-by-pixel subtraction to generate a difference image. X9 and X10 reformat the images for output.

4.6 Discussion

The implementations described above compare favorably with a custom VLSI design developed at Sarnoff Research Center [vand92]. This chip, known as PYR, can produce both a Gaussian and Laplacian pyramid for a 512 x 480 image in 22.7 ms, using a clock rate of 15.7 MHz. The chip size is approximately 300x300 mil, and is packaged in an 84-pin PLCC.

The pyramid implementations described in this paper are *almost* as sophisticated. The SPLASH 2 designs are currently limited to a fixed input image size of 512×512 pixels, unlike the PYR chip, and the SPLASH 2 implementation is considerably larger. However, the processing speeds are comparable, particularly if a second SPLASH 2 board is used to accommodate both pyramid types at frame rate. Considering only arithmetic operations, this application performs in excess of 500 million operations per second. Example images that have been computed using SPLASH 2 are shown in Figure 9.

5 Summary

This paper has described the design and implementation of two nontrivial image-processing algorithms on the SPLASH 2 custom computing platform. The two algorithms -- the Hough transform and Gaussian/Laplacian pyramid generation -- are common in image analysis and are computationally intensive. Both may be described as mappings from an input image to data structures containing new images.

Gaussian pyramid generation has been designed for real-time operation at the standard video frame rate of 30 images/second. The processing rate for generating Hough arrays is more difficult to quantify. This is because Hough processing is linear in the number of edge points in the input image. The design given here can utilize a PE (consisting of one FPGA and one static RAM) that can be replicated over the number of SPLASH 2 boards that are available.

It is significant that SPLASH 2 can be configured to perform either task at performance levels that rival or equal those of dedicated, application-specific hardware. Furthermore, with a sufficient number of SPLASH 2 array boards, these two tasks (along with others) could be pipelined to operate concurrently.

Perhaps the major architectural limitation of SPLASH 2 for real-time image processing is the local FPGA-to-RAM interface. Since dual-port RAMs are not provided, a read-modify-write operation requires 4 clock cycles. This operation is used extensively in the designs discussed here. A secondary architectural issue is data routing. The crossbar offers great flexibility, but image-processing applications always seem to beg for slightly higher bandwidth than is available.

Other applications operational on the laboratory *VTSplash* system include:

- 2-D Fast Fourier Transform (using floating point).
- Expandable 8×8 convolver (online filter adjustment).
- Pan and zoom.
- Median filtering.
- Morphologic operators
- Histogram and graphical display.
- Region detection and labeling
- Profile classifier.

6 Acknowledgments

The entire VTSplash team at Virginia Tech contributed to the work described in this paper. In addition to the authors, the team includes Brad Fross, Jeff Nevits, Jim Peterson, Ramana Rachakonda, Nabeel Shirazi, and Adit Tarmaster. The authors are also indebted to Jeff Arnold and Duncan Buell of the Supercomputing Research Center, and to the Institute for Defense Analyses.

7 Bibliography

[Arno92] J. M. Arnold, D. A. Buell, and E. G. Davis, "Splash 2," *Proceedings: 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 316-322, 1992.

[Arno93] J. M. Arnold, "The Splash 2 Software Environment," *Proceedings: 1st IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 88-93, Apr. 1993.

[Burt83] P. J. Burt and E. H. Adelson, "The Laplacian Pyramid as a Compact Image Code," *IEEE Transactions on Communications*, vol. COM-31, no. 4, pp. 532-540, Apr. 1983.

[Duda72] R. O. Duda and P. E. Hart, "Use of the Hough Transform to Detect Lines and Curves in Pictures," *Communications of the ACM*, vol. 15, pp. 11-15, 1972.

[vand92] G. S. van der Wal and P. J. Burt, "A VLSI Pyramid Chip for Multiresolution Image Analysis," *International Journal of Computer Vision*, vol. 8, no. 3, pp. 177-189, 1992.

[Houg62] P. V. C. Hough, "A Method and Means for Recognizing Complex Patterns," US Pattent No. 3,069,654, 1962.

[Xili91] Xilinx, Inc., *The Programmable Gate Arrays Users Book*, San Jose, California, 1991.

Figure 8. Hough transform example: a) input image, b) iput image after edge detection, c) Hough representation of input image, and d) overlay of detected lines onto the input image.

Figure 9. *(a) Gaussian (low-pass) and (b) Laplacian (band-pass) pyramids. As one progresses toward higher levels of the Gaussian pyramid, increasingly larger image details are removed. The same is true for the Laplacian pyramid, except that edge information is represented. The top level of the Laplacian pyramid is missing in this example.*