A Graphics Hardware Implementation of the Generalized Hough Transform for fast Object Recognition, Scale, and 3D Pose Detection

Robert Strzodka

research center caesar
D-53111 Bonn
strzodka@caesar.de

Ivo Ihrke, Marcus Magnor

Max-Planck-Institut für Informatik
D-66123 Saarbrücken
{ihrke,magnor}@mpi-sb.mpg.de

Abstract

The generalized Hough transform constitutes a well-known approach to object recognition and pose detection. To attain reliable detection results, however, a very large number of candidate object poses and scale factors need to be considered. In this paper we employ an inexpensive, consumer-market graphics card as the "poor man's" parallel processing system. We describe the implementation of a fast and enhanced version of the generalized Hough transform on graphics hardware. Thanks to the high bandwidth of on-board texture memory, a single pose can be evaluated in less than 3 ms, independent of the number of edge pixels in the image. From known object geometry, our hardware-accelerated generalized Hough transform algorithm is capable of detecting an object's 3D pose, scale, and position in the image within less than one minute.

1. Introduction

Given the geometry of an object and an image containing this object, we want to identify its pose, scale and position in the image. The human visual system constantly performs this task with little effort, but it has proven very difficult for computers to perform equally well automatically. In the abscence of a-priori knowledge about the world, usually an exhaustive search of the image must be performed to identify the object.

The generalized Hough transform (GHT) is a technique to perform this search by discretizing all possible transformations between object and image space and testing them individually [2, 28]. The percentage of resulting matches between object and image features are interpreted as the likelihood for a respective transformation to be correct. However, to attain reliable GHT detection results, a large number of object features and a high-resolution Hough table is required [8], resulting in a huge number of candidate transforms that must be evaluated. Therefore, this exhaustive search over all possible object positions in the image, its different poses and (within a preset range) varying size is very time- and memory-consuming. An implementation on a modern PC takes in the order of half an hour to reliably detect an object in a image taken from a completely unknown perspective [18].

Since the Hough Transform involves many similar operations, parallel implementations have been often used to improve its performance. In particular, distributed memory machines [19, 9], pyramid multiprocessors [26, 1], reconfigurable architectures [21, 20] and special purpose hardware [5] have been applied to reduce computation time. Most hardware implementations focus on the Hough transform itself, but many aspects of parallelization can also be applied to the generalized Hough transform [17, 3].

While many parallel architectures have already been used for implementing the GHT, we have adapted it to graphics hardware, because graphics hardware has the great advantage that its functionality comes for free. The graphics cards shipped with modern PCs are already powerful enough to be used for parallel computing, but even if one opts for high-end cards to maximize performance, the cost of purchase and operating this kind of parallel co-processor is very little in comparison to the dedicated hardware architectures listed above. Therefore, the use of graphics hardware for computations more complex than advanced procedural texturing and shading [22, 23] is steadily gaining popularity: Wavelet transforms [14], morphological operations [13], computation of Voronoi diagrams [12], volume rendering [6], ray tracing [24], flow visualization [11, 15, 30], segmentation [25], robot motion planning [16], and artificial neural networks [4] have already been implemented in graphics hardware. Even discrete solvers for partial differential equations in graphics hardware have been described [10] and also successfully applied

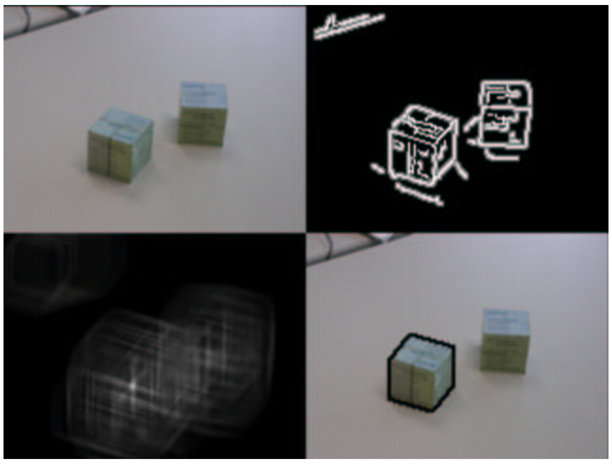

Figure 1. Object recognition using the generalized Hough transform: First edge pixels are detected in the real-world image. Then the edge image is convolved with pre-computed object outlines. The object location with the strongest signal indicates the best-matching object position in the image. By comparing the signal peaks from all object outlines, a ranking of the most probable object poses, sizes and image positions is established.

to various problems. In all these applications the restricted precision of the number formats in graphics hardware is seen as a major problem. We cope with this problem by applying a simulated extended precision format (Section 3).

The rest of the paper has the following structure. In the next Section 2, the GHT variant used here is outlined. Section 3 describes the graphics hardware implementation details of the presented system. Performance is evaluated and detection results are presented in Section 4. Prospective applications of the system and further improvements are addressed in the final Section 5.

2. Object Localization and Pose Detection

In this section, we re-phrase the GHT in a form most suitable for the implementation on graphics hardware.

For known 3D object geometry, we pre-compute the poses by rendering the object from various camera positions. During this process, we take into account object symmetry to obtain an even discretization of the parameter space without unnecessary repetitions and to minimize the number of poses.

After rendering, each pose k is transformed into a list l^k of 2D vectors of edge pixel positions. Each vector $l^{k,i}$ in this list represents the offset from an edge pixel to the center of the rendered object pose. The length $|l^k|$ of the list corresponds to the number of edge pixels of the given pose k.

To identify object edges in the image of a natural scene (Fig.1 upper left), the Canny Edge detector is applied [7]. The resulting edge image I^{edge} is traversed, and edge pixels are assigned a certain value $v(0)$. Other pixels are assigned values $v(d), d > 0$ depending on their distance d to an edge (Fig.1 upper right) where v is a monotoneously decreasing function. Although this dilation of edges slightly reduces the accuracy of object location, it greatly reduces the number of necessary poses for finding appropriate pose candidates. Moreover, using graphics hardware the edge dilation does not increase the execution time for the convolution below.

To find an object's position and pose in the image, the edge image is convolved with the previously generated offset list l^k of the objects' outlines:

$$\bar{C}^k(x,y) := \sum_{i=1}^{|l^k|} I^{\text{edge}}(x + l_x^{k,i}, y + l_y^{k,i}) \quad (1)$$

$$C^k(x,y) := f\left(\bar{C}^k(x,y), |l^k|\right) \quad (2)$$

where $f(a,b) := a/b$ simply normalizes the result against the length of the current object outline since $|l^k|$ is the number of edge pixels in the pose k. Note that f could easily incorporate a non-linear weighting function of the votes, as might be motivated by the human visual system. $\bar{C}^k(x,y)$ is the sum of the votes at position (x,y) for pose k, and $C^k(x,y)$ is the normalized vote used for the identification of maxima.

Given the normalized votes for each pose k, we can compute the most probable pose $M_{\text{pose}}(x,y)$ with the highest vote $M_{\text{conv}}(x,y)$ at position (x,y):

$$\begin{aligned} M_{\text{conv}}(x,y) &:= \max\left\{C^{k(x,y)} | k = 1 \ldots N^{\text{pose}}\right\} \\ M_{\text{pose}}(x,y) &:= \arg_k \max\left\{C^k(x,y) | k = 1 \ldots N^{\text{pose}}\right\} \\ M(x,y) &:= \left(M_{\text{conv}}(x,y), M_{\text{pose}}(x,y)\right) \quad (3) \end{aligned}$$

If desired, one could also compute a list of the n most probable values at each position, at the cost of higher computational complexity. But as long as the same object does not appear multiple times centered around the same image position, it completely suffices to determine only the most probable pose at each pixel position.

From M we obtain the most probable positions of the object under consideration by extracting the J largest values $M_{\text{conv}}(x_j, y_j), 1 \le j \le J$ and the corresponding most probable poses $M_{\text{pose}}(x_j, y_j), 1 \le j \le J$, where J can be a very large number, since we have a candidate for each position in the image. The first J maxima can then be used to initialize a decision process between the corresponding transformations. A simple approach would be to further discretize the orientation, scale, and position of the poses around the detected parameters and then restart the convolution with new offset lists l^k. One could even restrict the convolution to bounding boxes, depending on the detected scale and image positions of the J best poses. But once the huge parameter space is reduced to few candidates, more sophisticated approaches exist to decide between these candidates [29]. Such algorithms, however, include many conditional statements and an irregular memory access pattern which render them far less suitable for massive parallelization. Therefore, we have restricted our graphics hardware implementation to the first computationally most demanding part of estimating the J most probable poses among tens of thousands. In fact, it is a very different problem estimating the parameters of the most probable poses among many than to select the most appropriate pose among a few. In the latter case much more computation time can be alloted to each transformation candidate to evaluate its correctness.

3. Graphics Hardware Implementation

The huge advantage of graphics hardware is its massive parallel computing power in comparison to its low price. To exploit the full potential of graphics hardware, however, it is crucial to avoid frequent changes in the graphics pipeline as well as to minimize data transfer between the graphics card and main memory. Especially the outstanding texture fill rate offers great potential for hardware acceleration if texture operations can be suitably utilized. Our implementation focuses on exploiting this potential for the most demanding part of the GHT, namely, the convolution between image edges and object outlines.

The algorithm starts with the initialization which generates the lists of offset vectors l^k for each pose k by rendering the 3D geometry of the object with the associated Eulerian angles, object rotation and scale, and extracting the offset vectors $l^{k,i}$ from the read-back image. Then the software generated enhanced edge image I^{edge} is stored in a 2D texture. After this initialization, all computations take place on the graphics hardware without data transfer to or from the main memory.

The convolution with pose k (Eq. 1) is performed by repeatedly drawing $l^{k,i}$ translated versions of I^{edge} into the frame buffer while adding the rendered images with an additive blending mode. Using the graphics hardware's blending feature has the advantage that we do not need to explicitly access the intermediate sum in the frame buffer to perform the summation. But since the texture fill rate is usually much higher than the pixel fill rate, further acceleration can be achieved by multi-texturing each pixel with different translations of I^{edge}, whereby the texture coordinates account for the relative translation of one to another, and the texture environments perform the addition to a single color before the blending occurs. We perform a composite translation both in the frame buffer and the texture coordinates, because translations in the frame buffer coordinates often save bandwidth when we clip the translated source to the original image size and position. This is legitimate since we are not interested in objects whose centers lie outside of the original camera image. After all offsets from l^k have been considered, the result is stored in the texture $\bar{C}^k$.

In the previous section, we explained that edges in I^{edge} are assigned a certain value $v(0)$, and neighboring pixel values depend on their distance to an edge. A large value $v(0)$ has the advantage that many gradations can be encoded. However, the addition of many high values during the convolution can lead to an undesired overflow or, in the case of graphics hardware, rather to saturation. If we assume a resolution of 8 bits per color channel, then $v(0) = 1$ allows for 255 offsets to be savely added, yet without weight gradation. Assuming $v(0) = 2$, 127 offsets can be savely added, while edge neighbors can be assigned the value $v(1) = 1$. On future graphics hardware with 16-bit and 32-bit number formats, this limitations will become obsolete. Currently, in order to be able to use large outlines and fine weight gradation, we have implemented an unsigned version of the virtual 16-bit fixed-point format for RGBA8 textures [27], which emulates 16-bit precise operations on DX8 graphics hardware. This costs a factor of about two due to the doubled amount of data to be transfered, and an additional factor of two due to the need of re-transferring each intermediate sum to texture memory, since the 16-bit emulation cannot be achieved by blending alone.

The normalization of the transform result (Eq. 2) takes place as a 2-dimensional dependent texture access into a sufficiently large pre-computed look-up texture f, which represents a discrete 2-dimensional function and delivers an encoded 16-bit fixed-point result. As even the large memory of current graphics cards cannot hold all the textures C^k (usually $k > 10000$), the computation of the maximum is performed regularly as soon as a certain number of poses have been computed. By clustering results in a frame buffer of maximal size we may compute convolutions with more than 100 poses before switching to the maximum computation, thus reducing changes in the graphics pipeline to a minimum. During the pixelwise maximum evaluation, the texture M holds the current maximum and the index of the corresponding pose at each image position (x, y). For each k, the texture M is updated through

$$\begin{aligned} M(x,y) \quad &= \quad (M_{\text{conv}}(x,y) > C^k(x,y))? \\ &\quad\quad M(x,y) : \left(C^k(x,y), k\right) ; \end{aligned} \tag{4}$$

at each position (x, y). Both the normalization of $\bar{C}^k$ to C^k (Eq. 2) and the maximum computation (Eq. 3) make use of the virtual 16-bit format, even when the convolution uses

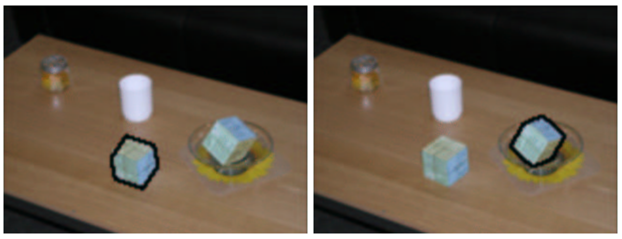

Figure 2. The 1st and 6th most probable cube poses in this table scene.

only 8 bits, because only the additional precision ensures accurate comparisons (Eq. 4) of the normalized convolution results. In contrast to the 16-bit usage in the convolution, the 16-bit precise normalization and maximum evaluation do not cost significant performance, since these steps together consume less than 5% of the overall time.

M is a RGBA8 texture where the convolution result M_{conv} and the pose index M_{pose} are stored as 16-bit fixed-point numbers represented by two 8-bit color channels each. In this way we may index up to $2^{16} = 65536$ different poses. The update (Eq. 4) itself can be implemented in a way very similar to the subtraction in the virtual 16-bit format [27].

After all poses l^k have been processed, the texture M contains at each (x, y) the index of the most probable pose for this location. This final result is transferred to main memory, where the J largest values and pose indices are extracted from M. Here we see the great advantage in the use of the virtual 16-bit format. It allows us to perform the entire process in graphics hardware, and we need to transfer only a single condensed image result back to the main memory, while usually each of the convolution results ($k > 10000$) would have to be transmitted, leading to bottle neck of memory bandwidth.

4. Results

We tested our algorithm with simple objects, which we had previously measured and modeled by hand. These models were then used to recognize the objects in different scenes (Figs. 2 to 5). In all these examples perspective distortion is assumed to be negligible i.e. a camera with long focal length is used. Symmetries of the objects are used to reduce the number of generated poses. We tested two versions of the algorithm (see Sec. 3), one with 8-bit and one with 16-bit precise convolution (Eq. 1). They differ in their ability to handle weight gradation around the edges. While the 8-bit version can handle at most two levels reasonably, the additional eight bits of the 16-bit version can be used to encode up to 256 different values in the edge image. Figure

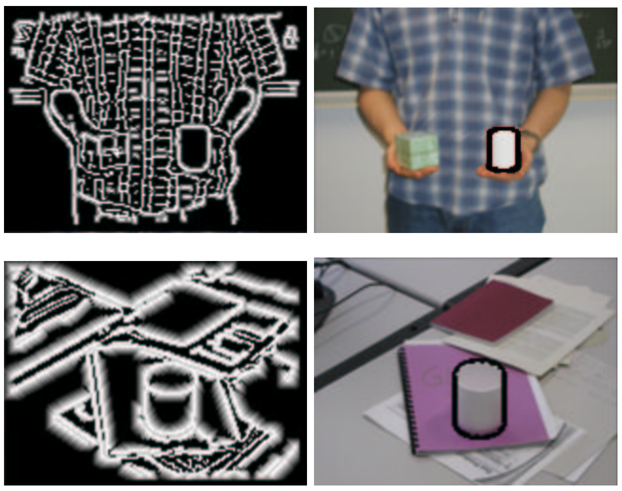

Figure 3. Upper image: Edge image with one pixel dilation as used for the 8-bit version of the algorithm, lower image: edge image with a 5 pixel dilation as used by the 16-bit version, decrease of the weights is inversely linear.

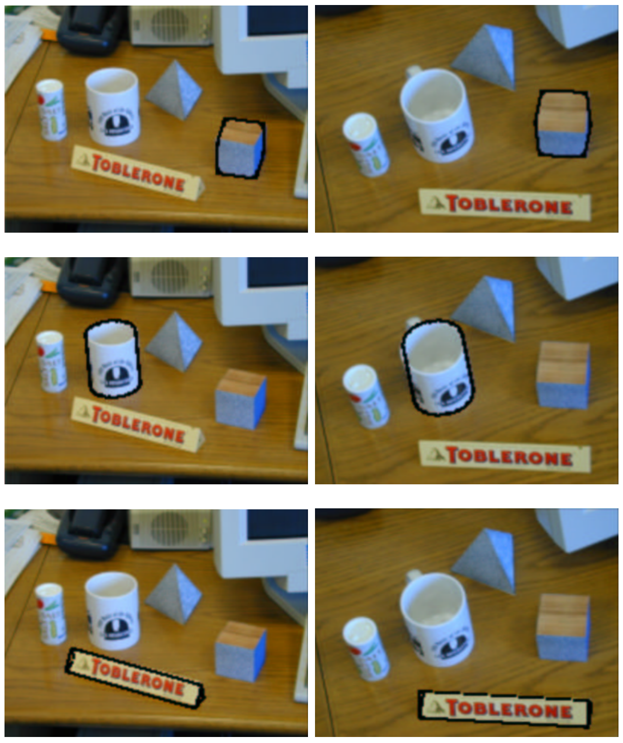

Figure 4. Detection of different objects in two views of the same scene using the 16-bit version of the algorithm . The results for the 8-bit version are very similar. Run-times are given in Tables 1 and 2.

Object	Image	Image Size	Num. of Poses	Num. of Scales	Num. of Offsets	Init. Time (sec)	Conv. Time (sec)	Operations per second
cube	Figure 4: left	200×150	725	20	1409622	14.27	26.20	$1.61 * 10^9$
cube	Figure 4: right	200×150	725	20	1409622	14.16	25.64	$1.65 * 10^9$
cup	Figure 4: left	200×150	713	30	3192192	71.53	49.20	$1.95 * 10^9$
cup	Figure 4: right	200×150	713	30	3192192	65.53	47.92	$2.00 * 10^9$
toblerone	Figure 4: left	200×150	504	35	3370871	191.75	46.94	$2.00 * 10^9$
toblerone	Figure 4: right	200×150	504	35	3370871	196.16	45.16	$2.22 * 10^9$

Table 1. Run-time data for the examples depicted in Fig. 4 for the 8-bit version of the algorithm

Object	Image	Image Size	Num. of Poses	Num. of Scales	Num. of Offsets	Init. Time (sec)	Conv. Time (sec)	Operations per second
cube	Figure 4: left	200×150	180	5	86711	1.31	5.17	$5.03 * 10^8$
cube	Figure 4: right	200×150	180	5	86711	1.27	5.16	$5.05 * 10^8$
cup	Figure 4: left	200×150	171	7	173869	3.86	9.38	$5.56 * 10^8$
cup	Figure 4: right	200×150	789	7	801418	16.73	43.08^*	$5.58 * 10^8$
toblerone	Figure 4: left	200×150	124	8	187736	11.27	9.39	$6.00 * 10^8$
toblerone	Figure 4: right	200×150	551	8	830568	48.27	41.47^*	$6.01 * 10^8$

Table 2. Run-time data for the examples depicted in Fig. 4 for the 16-bit version of the algorithm. (*) Due to poor pose estimation for this object, a similar pose discretization as in Table 1 had to be used, resulting in more overall poses to be tested and thus longer computation time.

3 shows the differences between the enhanced edge images used for the 8-bit (upper) and 16-bit (lower) version of the algorithm.

Figure 2 illustrates the ability of the algorithm to deal with the same object appearing more than once in the image. The outlines of identified, dissimilar cubes among the ten best matches are overlaid. Finally Figure 4 depicts the results of running the algorithm on two views of an identical scene, identifying different objects in the scene. While both versions of the algorithm produce similar results their run-time behaviour differs as can be seen in Tables 1 and 2. The 8-bit version executes each convolution faster as explained in Section 3, but needs to be run with a finer pose discretization and a larger number of scales in order to produce reliable results. The 16-bit version on the other hand is able to handle nearly arbitrary amounts of dilation. A trade-off between accuracy of the estimated pose and position versus computation time can thus be established.

Table 1 shows the run-time data for the 8-bit version of the algorithm. The algorithm was run with the same parameters on the left and the right image of figure 4. It can be noted that run-time is nearly linear depending on the overall number of offsets. The deviation from linear increase is due to a per pose setup step, which includes the change of a texture environment. Thus run-time for poses with a large number of offsets scale almost ideally. The much larger initialization times of “cup” and “toblerone” with respect to “cube” are explained by our unoptimized rendering of the objects silhouettes whith quadratic increase of initialization time with increasing object size Table 2 shows the corresponding values for the 16-bit version of the algorithm. Despite being computationally more expensive, the algorithm performs faster, because it is able to give comparable results using far less poses and scales. The disadvantage is the degraded accuracy in the localization of poses. In fact the poses for the object-image pairs marked with (*) were poor, so that the detection was rerun with a pose discretization comparable to the one used in Table 1. This resulted in detections similar to the 8-bit algorithm, at approximately the same computational costs.

All measurements were performed on an nVidia GeForce 4 graphics card. For optimal performance of the algorithm, pose discretization should be chosen such that poses differ less than the amount of dilation applied to the edge image, otherwise objects could be missed. It is also helpful to apply an upper and lower bound for the apparent size of the object to reduce the number of scales to be searched for.

When compared to the original software implementation [18] of the described GHT-based object recognition scheme, graphics hardware accelerates the algorithm’s performance by a factor of about 10.

5. Conclusions

A graphics hardware-accelerated implementation of an efficient generalized Hough transform variant has been presented. It is able to perform an exhaustive search among many thousands of object poses and different object sizes in less than one minute. The described approach's main advantage is its capability to quickly and autonomously return high-probability object-to-image transformations with only a conservative guess of object size range as initial search parameter. A short list of most-probable object positions, 3D poses, and sizes is extracted that can then be used in a refinement step to pinpoint the correct object parameters. Moreover, since the algorithm always obtains the most probable pose for each image position, the parameter space is so dramatically reduced, that more elaborate techniques could be used to identify objects in cases when the highest convolution results do not match the object. Therefore, in its presented form, the algorithm is intended to provide a fast parameter space reduction which either directly estimates the poses or initializes a refined search with other methods.

References

[1] M. Atiquzzaman. Pipelined implementation of the multiresolution hough transform in a pyramid multiprocessor. *PRL*, 15(9):841–851, September 1994.

[2] D. Ballard. Generalized hough transform to detect arbitrary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(2):111–122, 1981.

[3] D. Baumann and S. Ranka. The generalized hough transform on an mimd machine. *Journal of Undergraduate Research in High-Performance Computing*, 2, 1992.

[4] C.-A. Bohn. Kohonen feature mapping through graphics hardware. In *Proceedings of Int. Conf. on Compu. Intelligence and Neurosciences 1998*, 1998.

[5] J. Bruguera, N. Guil, T. Lang, J. Villalba, and E. Zapata. Cordic based parallel/pipelined architecture for the hough transform, 1996.

[6] B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In A. Kaufman and W. Krueger, editors, *1994 Symposium on Volume Visualization*, pages 91–98. ACM SIGGRAPH, 1994. ISBN 0-89791-741-3.

[7] F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[8] W. E. L. Grimson and D. P. Huttenlocher. On the sensitivity of the hough transform for object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-12(3):255–274, 1990.

[9] N. Guil and E. L. Zapata. A parallel pipelined hough transform. In *Euro-Par, Vol. II*, pages 131–138, 1996.

[10] M. J. Harris, G. Coombe, T. Scheuermann, and A. Lastra. Physically-based visual simulation on graphics hardware. In *Proceedings of Graphics Hardware 2002*, pages 109–118, 2002.

[11] W. Heidrich, R. Westermann, H.-P. Seidel, and T. Ertl. Applications of pixel textures in visualization and realistic image synthesis. In *ACM Symposium on Interactive 3D Graphics*. ACM/Siggraph, 1999.

[12] K. E. Hoff III, J. Keyser, M. Lin, D. Manocha, and T. Culver. Fast computation of generalized Voronoi diagrams using graphics hardware. *Computer Graphics*, 33(Annual Conference Series):277–286, 1999.

[13] M. Hopf and T. Ertl. Accelerating Morphological Analysis with Graphics Hardware. In *Workshop on Vision, Modelling, and Visualization VMV '00*, pages 337–345, 2000.

[14] M. Hopf and T. Ertl. Hardware Accelerated Wavelet Transformations. In *Proceedings of EG/IEEE TCVG Symposium on Visualization VisSym '00*, pages 93–103, 2000.

[15] B. Jobard, G. Erlebacher, and M. Y. Hussaini. Hardware-accelerated texture advection for unsteady flow visualization. In *Visualization '00*, pages 155–162, 2000.

[16] J. Lengyel, M. Reichert, B. Donald, and D. Greenberg. Real-time robot motion planning using rasterizing computer graphics hardware. In *Proceedings of SIGGRAPH 1990*, pages 327–335, 1990.

[17] Z. Li, B. Yao, and F. Tong. A linear generalized hough transform and its parallel implementation. *CVPR*, pages 672–673, 1991.

[18] M. Magnor. Geometry-based automatic object localization and 3-d pose detection. *Proc. IEEE Southwest Symposium on Image Analysis and Interpretation (SSIAI-2002)*, Santa Fe, USA, pages 144–147, Apr. 2002.

[19] A. U. Mohammed. Performance of the hough transform on a distributed memory multiprocessor. *Pattern Recognition Letters*, 15(9), 1994.

[20] Pan, Li, and Hamdi. An improved constant-time algorithm for computing the radon and hough transforms on a reconfigurable mesh. *IEEETSMC: IEEE Transactions on Systems, Man, and Cybernetics*, 29, 1999.

[21] Pavel and Akl. Efficient algorithms for the hough transform on arrays with reconfigurable optical buses. In *IPPS: 10th International Parallel Processing Symposium*. IEEE Computer Society Press, 1996.

[22] M. S. Peercy, M. Olano, J. Airey, and P. J. Ungar. Interactive multi-pass programmable shading. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings,*, Annual Conference Series, pages 425–432. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[23] K. Proudfoot, W. R. Mark, S. Tzvetkov, and P. Hanrahan. A real-time procedural shading system for programmable graphics. In E. Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, Annual Conference Series, pages 159–170. ACM Press / ACM SIGGRAPH, 2001.

[24] T. Purcell, I. Buck, W. Mark, and P.Hanrahan. Ray tracing on programmable graphics hardware. In *Proceeding of SIGGRAPH 2002*. ACM, 2002.

[25] M. Rumpf and R. Strzodka. Level set segmentation in graphics hardware. In *Proceedings ICIP'01*, volume 3, pages 1103–1106, 2001.

[26] S. Shoari, A. Kavianpour, and N. Bagherzadeh. Pyramid simulation of image-processing applications. *IVC*, 12(8):523–529, October 1994.

[27] R. Strzodka. Virtual 16 bit precise operations on RGBA8 textures. In *Proceedings VMV'02*, 2002.

[28] R. Veltkamp. Shape matching: Similarity measures and algorithms, 2001. Technical Report UU-CS-2001-03, Utrecht University, the Netherlands.

[29] R. Veltkamp and M. Hagedoorn. State-of-the-art in shape matching, 1999. Technical Report UU-CS-1999-27, Utrecht University, the Netherlands.

[30] D. Weiskopf, M. Hopf, and T. Ertl. Hardware-accelerated visualization of time-varying 2d and 3d vector fields by texture advection via programmable per-pixel operations. In *Proceedings of VMV'01*, pages 439–446, 2001.