An Adaptive On-Line Evolutionary Visual System

Marc Ebner
Eberhard-Karls-Universität Tübingen
Wilhelm-Schickard-Institut für Informatik
Abt. Rechnerarchitektur, Sand 1, 72076 Tübingen
marc.ebner@wsii.uni-tuebingen.de

Abstract

In evolutionary computer vision, algorithms are usually evolved which address one particular computer vision problem. Quite often, a set of training images is used to evolve an algorithm. Another set of images is then used to evaluate the performance of those algorithms. In contrast of this standard form of algorithm evolution, it is proposed to develop a vision system which continuously evolves algorithms based on the task at hand. This adaptation of computer vision algorithms would happen on-line for every image which is presented to the system. Such a system would continuously adapt to new environmental conditions.

1. Introduction and Motivation

Today, vision systems are in use in many places. They are used on autonomous mobile robots or as stationary surveillance systems. We propose to make such systems adaptive to their environment using evolutionary algorithms.

In current work on evolutionary computer vision [1], one usually tries to solve a given computer vision problem using an evolutionary algorithm. Evolutionary algorithms use simulated evolution to search the space of possible solutions for a method which solves the given problem [2]. An evolutionary algorithm works with a population of individuals. Each individual represents a possible solution for the given problem. The possible solutions are found by essentially using the same operators which are used by natural evolution: reproduction, variation and selection.

Suppose that we are looking for an algorithm which extracts some kind of information from an image, e.g. where an object with a known shape is located within the image. In evolutionary computer vision, one often works with two sets of images. One set of images is used to train the algorithm, i.e. this set is used during evolution in order to evaluate the performance of the individuals. A second set of images is used to see how well the evolved algorithm generalizes to previously unseen images.

Experiments in evolutionary computer vision are usually computationally very expensive. One is working with a population of individuals, i.e. possible solutions to the given problem. This population of individuals is evolved for several generations. In order to evaluate a single individual, it may be necessary to apply a sequence of image processing operators such as edge detection, computation of a histogram or transforming the image to frequency space [3], [4]. Evaluating several sequences of image processing operators over many generations will take a considerable amount of time. This is very expensive and hardly suited for on-line image processing.

With the advent of powerful computer graphics hardware, however, it may soon be possible to develop a true on-line evolutionary computer vision system which continuously processes the visual information and which would adapt itself to different environmental conditions. Such a system would be very different from traditional computer vision systems. Traditional computer vision systems are usually developed to complete a certain task within a standardized environment. Such systems are hardly adaptive and often break when run in a different environment. This happens quite frequently in the field of robotics especially when working with autonomous mobile service robots. Autonomous mobile service robots have to perform well in many different environments. The environments cannot be standardized. The robot may have to perform its task under many different lighting conditions. It has to recognize different objects irrespective whether there is little light available (and there is quite a lot of noise in the image data) or whether there is a sufficient amount of light available (and there is little noise in the data). Such vision systems would be context-aware pervasive systems.

Which algorithm is optimal for a given task almost always depends on the current environmental conditions. It is therefore proposed to create an on-line adaptive vision system. This adaptive system would continuously process the visual information received from the camera. The system would search for alternative algorithms in the background on the visual input received by the system. Multiple algorithms would be explored in parallel and the best one would replace the current algorithm in use. The system would therefore continuously adapt to the incoming data in a way that the performance is always the best possible performance independent of any environmental conditions, i.e. the system would perform optimally when the environment is brightly lit and would also perform optimally when little light is

available.

2. Evolutionary Computer Vision

In evolutionary computer vision, one tries to evolve or optimize an algorithm which solves a given computer vision problem [1]. Work in evolutionary computer vision started in the 1990s. One of the first papers advocating the idea is probably a paper by Lohman [5]. He used an Evolution Strategy to compute the Euler number of an image [6]. In the 1990s, evolutionary algorithms were used to evolve low-level operators such as edge detectors [7], feature detectors [8], or interest point detectors [9]. It was already clear that in principle, evolutionary computer vision may be used to evolve fully adaptive operators (see Ebner and Zell [10]) which adapt themselves to the task at hand and that genetic programming would be useful to image analysis [11]. For instance, Johnson et al. [12] used genetic programming very successfully to evolve visual routines. Evolutionary computer vision is a very active research area ranging from evolution of low-level detectors [13], to object recognition [14] or camera calibration [15]. Latest research on evolutionary computer vision is published in the European Workshop on Evolutionary Computation in Image Analysis and Signal Processing organized annually by Stefano Cagnoni.

Unfortunately, currently, due to the enormous computational requirements, experiments in evolutionary computer vision have to be performed off-line. The systems are used to evolve an operator or method to achieve some dedicated task. Only then the result can be used in real time. An outline of an evolutionary computer vision system is shown in Fig. 1. In some cases, an algorithm exists but it may not be optimal yet. If this is the case, one would use Genetic Algorithms [16] or Evolution Strategies [17] for parameter optimization. One works with a population of individuals. Each individual represents a possible solution to the given problem, i.e. the parameters of the problem are coded inside the individual. When Genetic Algorithms are used, parameters are coded using bit-strings. When Evolution Strategies are used, parameters are coded using floating-point vectors.

Darwin's principle "survival of the fittest" is used to select only those individuals from the population which are better than their peers at solving the given problem. Excellent individuals have a much higher probability of being selected than bad individuals. The best individuals are allowed to reproduce into the next generation. The offspring are usually not identical to their parents. They are modified slighly using genetic operators such as mutation and crossover. The genetic material, i.e. the parameters of the given problem, are exchanged (one could also say shuffled) between two or possibly more parents when crossover is used. This shuffled material then represents the offspring. The genetic material is modified by applying the mutation operator. The mutation

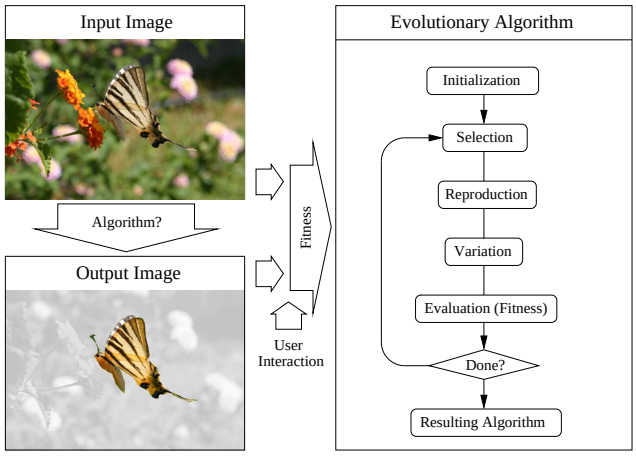

Figure 1. An evolutionary computer vision system.

operator usually changes a single parameter slightly. Some individuals may be produced using only crossover, some individuals may be produced using only mutation, some may be produced using both. When a new population is completely filled, the process is repeated for another generation for a generational based evolutionary algorithm. This process continues until a good enough solution is found.

Each individual is evaluated using a fitness function or error measure. The fitness function or error measure basically describes how good the individual is at solving the given problem. Suppose that we want to optimize an object detection algorithm. The algorithm should return 1 for each pixel where the object is located and 0 everywhere else. Given such a problem, one could use a simple distance function to evaluate the individuals. The distance function would compute the difference or error between the desired output of the object detection algorithm and the actual output of the object detection algorithm. A perfect individual would have an error measure of zero.

Apart from parameter optimization of existing algorithms, it is of course also highly desirable to be able to evolve an entire computer vision algorithm from scratch. One basically wants to answer the following question: In what order and with what parameters should the different operators which are known from the computer vision literature be applied to an input image in order to achieve a desired output. For such problems, Genetic Programming is highly applicable. Genetic Programming is used to automatically search the space of computer programs to evolve a solution to a given task [18], [19]. The instructions of the computer program correspond to the computer vision operators known from the literature.

Three different representations are widely in use for Genetic Programming. The first is a tree-based representation for the individuals where the nodes of the tree are the instructions of the program [20]. Genetic operators such as

mutation and crossover manipulate the structure of the tree. The mutation operator randomly selects a node from the tree and then replaces this node with a newly generated sub-tree. The crossover operator randomly exchanges the genetic material contained in two randomly chosen sub-trees from the two individuals.

The second representation is a linear sequence of instructions [21]. This variant is called linear Genetic Programming. The instructions of the program are operating on registers. Input data is supplied to the program via an input register and the output of the program is read out from an output register. Genetic operators such as mutation and crossover manipulate the linear sequence of instructions. If this type of representation is used for evolutionary computer vision, then the registers are thought to represent the entire image. The individual instructions or computer vision operators transform one image to another image.

A third representation, known as Cartesian Genetic Programming works with a two-dimensional grid of operators [22]. Operators of any given column may process the input from the operators positioned inside any previous column. Fig. 2 shows how interest points can be extracted from an image using a representation of the Cartesian Genetic Programming paradigm. First, horizontal and vertical edges are extracted from the input image. The derivative in x and y directions is squared and subtracted from the product of the second derivative in x and y directions. Finally, non-local maxima are suppressed.

Evolutionary computer vision systems are mostly used off-line to process the available visual data and to evolve an optimal algorithm or to optimize a given algorithm. These systems are not truly adaptive in a sense where they would immediately adjust their behavior to a new environment (i.e. to new visual data). In the next section, such an adaptive evolutionary vision system is described.

3. An Adaptive On-Line Evolutionary Visual System

In developing vision systems, especially in the field of robotics, it is very important to note that these systems must always be ready to process the available information. They continuously need to process every input image coming into the system. For some systems, data is gathered and after a certain amount of time has passed, the system is stopped and the available data is output, e.g. an environment model of the surrounding of the robot. That's not what is desired here. It is assumed here that the robot is always on and continuously processing the data.

Let us assume for a moment that we have some kind of computer vision algorithm which performs a desired task on the robot. That task could be anything from object detection (where the robot only visually recognizes a known object in its visual field) to complex tasks including object following

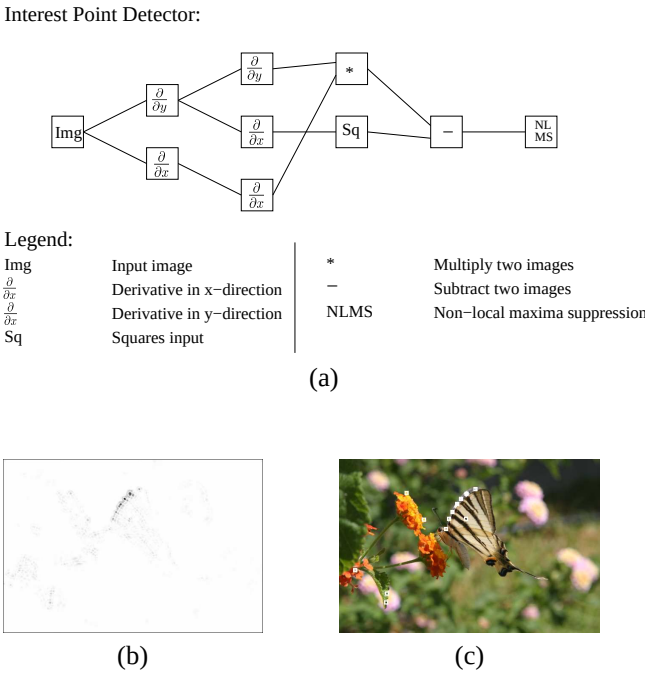

Figure 2. (a) Interest point detector. (b) Image computed by the interest point detector before suppression of non-local maxima. (c) Interest points overlayed on the original image.

(where the robot also has to follow the object and maintain the object within his field of view).

If our computer vision system is sufficiently powerful, then the system will be able to test alternative algorithms in the background. In other words, for every input image, the system would run multiple algorithms on the same input data. The algorithm performing best would win this round (Darwinian selection). The output of the winning algorithm would then be used to steer the robot or to present its findings to the user via a user interface. The system would then create slight variations of the winning algorithm which are evaluated using the next input data. Such a computer vision system is shown in Fig. 3. For this example, the winning algorithm would be algorithm no. 2 (marked in bold) because it extracts more of the wings of the butterfly.

The difficult part in developing such systems (apart from handling the enormous computational requirements) is to find objective criteria with which we can evaluate the different algorithms. Suppose that we would like to extract an object from an image, e.g. a ball. How would we evaluate algorithms for extracting that object? Given two algorithms which both extract some object from an image (we do not really know what it is) the following criteria would surely apply.

- An algorithm classifying more object pixels as object pixels compared to another algorithm should be preferred.

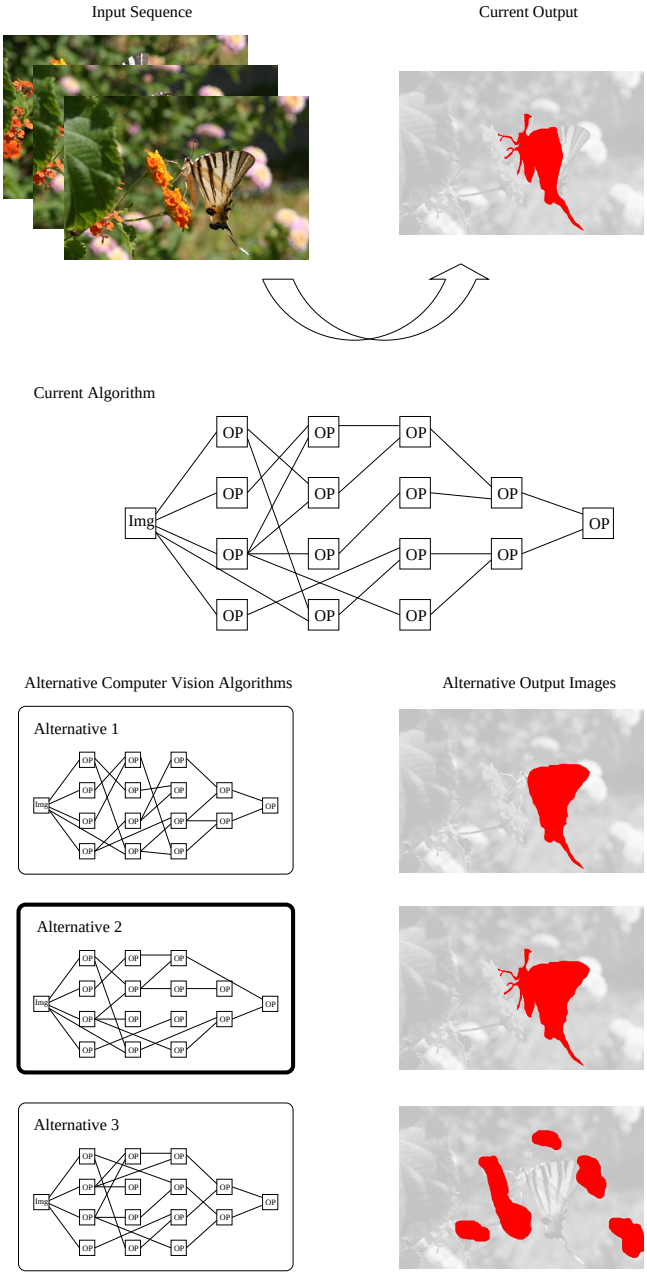

Figure 3. An adaptive on-line evolutionary visual system.

- An algorithm classifying more background pixels as background pixels compared to another algorithm should be preferred.

However, we cannot be sure what is an object pixel and what is a background pixel without using some high level ground truth knowledge for this evaluation. Another criteria for evaluating the algorithms would be taking the shape of the object into account. For a ball, we would know that the outline of the ball has to be round if the ball is completely contained inside the image and it is not occluded. In other words, an algorithm to extract a ball from the input sequence is a good algorithm if it always extracts some round object. The edges found at the outline of the extracted object must have the correct orientation. This could further be confirmed by looking at the shading information of the object. Texture would be another cue which could be used to discern one object from another.

Suppose that we would evaluate 5 alternatives for every input image. If we are able to work at a frame rate of 25Hz, that would mean that 125 alternative computer vision algorithms are evaluated per second or a total of 7500 algorithms per minute. The question is, how can we achieve such high frame rates as the application of computer vision operators is quite costly? We will see in the next section how such a system may be implemented. It is suggested to use the GPU to carry out the image processing operations. Given that modern computer graphics cards are equipped with high speed graphic processors, it makes sense to evaluate the algorithms directly on the graphics card.

4. Performing Image Processing Operations on the GPU

In recent years, graphics processing hardware has become increasingly powerful. It is even used to perform computations which are completely unrelated to computer graphics or computer vision for that matter. So far, algorithms such as sorting, searching, solving of differential equations, matrix multiplication and the fast Fourier transform have been ported to the graphics processing unit (GPU). A detailed survey of general-purpose computation on graphics hardware is given by Owens et al. [23]. Applications include the simulation of reaction-diffusion equations, fluid dynamics, image segmentation, ray tracing or computing echoes of sound sources.

A number of software packages have appeared which facilitate the development of GPU accelerated algorithms. Buck et al. [24] have developed a system for general-purpose computation on programmable graphics hardware. They basically use the GPU as a streaming coprocessor. Nvidia have developed the Compute Unified Device Architecture (CUDA) [25] which makes it possible to use the GPU as a massively parallel computing device using a C like language.

Several researchers have used the GPU for implementing various computer vision algorithms. Fung et al. [26] noticed very early on the capabilities of modern graphics processing units and implemented a projective image registration algorithm on the GPU. Yang and Pollefeys [27] implemented a hierarchical correlation based stereo algorithm in a clever way very efficiently on the GPU. Fung and Mann [28] showed how to implement simple image operations such as blurring, down-sampling and computing derivatives and even a real-time projective camera motion tracking routine. An computer vision software, called OpenVIDIA, initiated

by Fung et al. [29], is available to utilize the power of the GPU.

Modern graphics hardware is highly optimized for rendering triangles [30], [31]. Information within a triangle is obtained by interpolating data from the vertices. Originally, the rendering pipeline was fixed. Apart from specifying some parameters which is used by the rendering pipeline, it could not be modified by the user. In 1999, this changed with the introduction of programmable stages [23]. Modern graphics hardware allows the user to specify small programs which are either executed per vertex or per pixel.

These programs are called vertex and pixel shaders respectively. Originally, the programs for the pixel and vertex shaders had to be written in a special kind of assembly language. The available commands were all crafted to performing all kinds of computations which may have to be performed within the graphics pipeline. Instructions for vector and matrix arithmetic were included as well as specific instructions which may be needed when computing the brightness or color of a pixel. In recent years, C-like languages appeared which made it significantly easier to write shaders for different graphics hardware. Examples include Cg [32], developed by Nvidia, and the OpenGL Shading Language (GLSL) [33], created by the OpenGL ARB. The C-like code is compiled by the graphics driver into the appropriate shader code for the GPU wherever it is executed.

When using the vertex and pixel shaders for tasks other than 3D graphics rendering, one still needs to think in the vertex/pixel shader computer graphics paradigm. Computer vision operators can be implemented by sending four vertices through the graphics pipeline. The four vertices specify the end points of the rectangular image. The input image itself is made available to the pixel shader as a texture.

A blur shader is especially simple to implement because it can make use of a so called mip mapping mechanism. It simply accesses a down-sampled version of the texture and maps that to the entire image. A gradient shader computes the differences between adjacent pixels of the texture and then sums up the squared differences over all three color channels using the function for computing the dot product. A Laplacian shader adds up the differences between the central and the surrounding pixels. The clipping is performed automatically.

When writing computer vision algorithms, usually several operators have to be applied in sequence. The output of one operator is used as input of another operator. Currently, one can apply only one operator at a time via the GPU. The output of this operator is obtained by rendering to a texture. When the next operator should be applied, all of the textures which were generated in that way, have to be presented to the shaders implementing the next operator. In other words, the GPU is invoked once for every operator and the results are copied in between. This is an unnecessary overhead slowing down the entire process. The bottleneck is the image transfer between the CPU and the GPU which currently still has to be performed. The GPU could be used much more efficiently, if it were possible to directly read from and write to multiple output textures. Unfortunately, currently, this is not possible. As soon that this feature becomes available (and I have no doubt that it will become available) we will be able to build a continuously adapting visual system on the GPU.

5. Conclusions

In developing computer vision applications, several operators have to be applied in sequence to perform a given task. When developing computer vision algorithms the programmer decides in what sequence these operators have to be applied. Given the power of current graphic cards, it may soon become possible to develop an adaptive vision system on a GPU which continuously processes the visual information and automatically adapts itself to changing environmental condition.

References

[1] M. Ebner, “Aktuelles schlagwort: Evolutionäre bildverarbeitung,” *Informatik-Spektrum*, vol. 31, no. 2, pp. 146–150, Apr. 2008.

[2] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. Cambridge, Massachusetts: The MIT Press, 1992.

[3] R. Jain, R. Kasturi, and B. G. Schunck, *Machine Vision*. New York: McGraw-Hill, Inc., 1995.

[4] L. G. Shapiro and G. C. Stockman, *Computer Vision*. New Jersey: Prentice Hall, 2001.

[5] R. Lohmann, “Selforganization by evolution strategy in visual systems,” in *Evolution and Optimization ’89*, H.-M. Voigt, H. Mühlenbein, and Hans-Paul-Schwefel, Eds. Akademie-Verlag, 1990, pp. 61–68.

[6] ——, “Bionische Verfahren zur Entwicklung visueller Systeme,” Ph.D. dissertation, Technische Universität Berlin, Fachbereich 10 Verfahrenstechnik und Energietechnik, 1991.

[7] M. Ebner, “On the evolution of edge detectors for robot vision using genetic programming,” in *Workshop SOAVE ’97 - Selbstorganisation von Adaptivem Verhalten, VDI Reihe 8 Nr. 663*, H.-M. Groß, Ed. Düsseldorf: VDI Verlag, 1997, pp. 127–134.

[8] ——, “Evolution of hierarchical translation-invariant feature detectors with an application to character recognition,” in *Mustererkennung 1997, 19. DAGM-Symposium Braunschweig, 15.-17. September 1997*, E. Paulus and F. M. Wahl, Eds. Berlin: Springer-Verlag, 1997, pp. 456–463.

[9] ——, "On the evolution of interest operators using genetic programming," in *Late Breaking Papers at EuroGP'98: the First European Workshop on Genetic Programming*, Riccardo Poli, W. B. Langdon, Marc Schoenauer, Terry Fogarty, and Wolfgang Banzhaf, Eds. Paris, France: The University of Birmingham, UK, 14-15 Apr. 1998, pp. 6–10.

[10] M. Ebner and A. Zell, "Evolving a task specific image operator," in *Joint Proceedings of the First European Workshops on Evolutionary Image Analysis, Signal Processing and Telecommunications (EvoIASP'99 and EuroEcTel'99), Göteborg, Sweden, 1999*, R. Poli, H.-M. Voigt, S. Cagnoni, D. Corne, G. D. Smith, and T. C. Fogarty, Eds. Berlin: Springer-Verlag, May 1999, pp. 74–89.

[11] R. Poli, "Genetic programming for image analysis," in *Genetic Programming 1996, Proceedings of the First Annual Conference, July 28-31, 1996, Stanford University*, J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, Eds. Cambridge, Massachusetts: The MIT Press, 1996, pp. 363–368.

[12] M. P. Johnson, P. Maes, and T. Darrell, "Evolving visual routines," in *Artificial Life IV, Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, R. A. Brooks and P. Maes, Eds. Cambridge, Massachusetts: The MIT Press, 1994, pp. 198–209.

[13] L. Trujillo and G. Olague, "Synthesis of interest point detectors through genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference 2006, Seattle, Washington, July 8-12*. ACM, 2006, pp. 887–894.

[14] A. Treptow and A. Zell, "Combining adaboost learning and evolutionary search to select features for real-time object detection," in *Proceedings of the IEEE Congress on Evolutionary Computation, Portland, Oregon*, vol. 2. IEEE, 2004, pp. 2107–2113.

[15] P. Heinemann, F. Streichert, F. Sehnke, and A. Zell, "Automatic calibration of camera to world mapping in robocup using evolutionary algorithms," in *Proceedings of the IEEE International Congress on Evolutionary Computation, San Francisco, CA*. IEEE, 2006, pp. 1316–1323.

[16] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, Massachusetts: The MIT Press, 1996.

[17] I. Rechenberg, *Evolutionsstrategie '94*. Stuttgart: frommann-holzboog, 1994.

[18] J. R. Koza, *Genetic Programming. On the Programming of Computers by Means of Natural Selection*. Cambridge, Massachusetts: The MIT Press, 1992.

[19] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming - An Introduction: On The Automatic Evolution of Computer Programs and Its Applications*. San Francisco, California: Morgan Kaufmann Publishers, 1998.

[20] J. R. Koza, "Artificial life: Spontaneous emergence of self-replicating and evolutionary self-improving computer programs," in *Artificial Life III: SFI Studies in the Sciences of Complexity Proc. Vol. XVII*, C. G. Langton, Ed. Reading, Massachusetts: Addison-Wesley, 1994, pp. 225–262.

[21] P. Nordin, "A compiling genetic programming system that directly manipulates the machine code," in *Advances in Genetic Programming*, K. E. Kinnear, Jr., Ed. Cambridge, Massachusetts: The MIT Press, 1994, pp. 311–331.

[22] J. F. Miller, "An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach," in *Proceedings of the Genetic and Evolutionary Computation Conference*, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds. San Francisco, California: Morgan Kaufmann, 1999, pp. 1135–1142.

[23] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," in *Eurographics 2005, State of the Art Reports*, Aug. 2005, pp. 21–51.

[24] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for gpus: Stream computing on graphics hardware," in *International Conference on Computer Graphics and Interactive Techniques (ACM SIGGRAPH)*, Aug. 2004, pp. 777–786.

[25] NVIDIA, *NVIDIA CUDA. Compute Unified Device Architecture. Version 1.1*, 2007.

[26] J. Fung, F. Tang, and S. Mann, "Mediated reality using computer graphics hardware for computer vision," in *Proceedings of the Sixth International Symposium on Wearable Computers*. ACM, 2002, pp. 83–89.

[27] R. Yang and M. Pollefeys, "Multi-resolution real-time stereo on commodity graphics hardware," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2003, pp. 211–218.

[28] J. Fung and S. Mann, "Computer vision signal processing on graphics processing units," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, 2004*, vol. 5. IEEE, 2004, pp. 93–96.

[29] J. Fung, S. Mann, and C. Aimone, "Openvidia: Parallel gpu computer vision," in *International Multimedia Conference. Proceedings of the 13th annual ACM international conference on Multimedia, Singapore*, vol. 5. ACM, 2005, pp. 849–852.

[30] T. Akenine-Möller and E. Haines, *Real-Time Rendering*, 2nd ed. Natick, Massachusetts: A K Peters, 2002.

[31] OpenGL Architecture Review Board, D. Shreiner, M. Woo, J. Neider, and T. Davis, *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 2*, 5th ed. Reading, Massachusetts: Addison-Wesley, 2006.

[32] R. Fernando and M. J. Kilgard, *The Cg Tutorial. The Definitive Guide to Programmable Real-Time Graphics*. Boston, Massachusetts: Addison-Wesley, 2003.

[33] R. J. Rost, *OpenGL Shading Language. With contributions by John M. Kessenich, Barthold Lichtenbelt, Hugh Malan, and Mike Weiblen*, 2nd ed. Upper Saddle River, NJ: Addison-Wesley, 2006.