Towards Automated Learning of Object Detectors

Marc Ebner

Eberhard-Karls-Universität Tübingen
Wilhelm-Schickard-Institut für Informatik
Abt. Rechnerarchitektur, Sand 1, 72076 Tübingen, Germany
marc.ebner@wsii.uni-tuebingen.de
http://www.ra.cs.uni-tuebingen.de/mitarb/ebner/welcome.html

Abstract. Recognizing arbitrary objects in images or video sequences is a difficult task for a computer vision system. We work towards automated learning of object detectors from video sequences (without user interaction). Our system uses object motion as an important cue to detect independently moving objects in the input sequence. The largest object is always taken as the teaching input, i.e. the object to be extracted. We use Cartesian Genetic Programming to evolve image processing routines which deliver the maximum output at the same position where the detected object is located. The graphics processor (GPU) is used to speed up the image processing. Our system is a step towards automated learning of object detectors.

1 Motivation

A human observer has no problems in identifying different objects in an image. How do humans learn to recognize different objects in an image? Enabling a computer vision system to perform this feat is a daunting task. However, we try to work towards this goal. An ideal computer vision system would be able to automatically learn different object detectors from scratch. It is obviously highly desirable to develop self-adapting and self-learning vision systems which work without human intervention [4]. We have developed an evolutionary computer vision system which is able to automatically generate object detectors without human intervention.

Our system is based on a previously developed evolutionary vision system using GPU accelerated image processing [6]. Input to the system is a continuous stream of images. Each input image is processed by several different computer vision algorithms. The best algorithm is used to supply the overall output, i.e. to detect objects in the input image. Evolutionary operators are used to generate new alternative algorithms. The original system required user interaction to identify the objects to be detected. We have extended this system such that no user interaction is required.

For humans, motion serves as an important cue to identify interesting objects. Our system detects differences between consecutive images in order to detect independently moving objects in the image. Each detected object is equipped with

a 2D motion model which describes the motion of the object on the screen [3]. This motion model is continuously updated based on the motion differences between two consecutive images. By directly transforming a sequence of difference images into a 2D motion model, the computational resources needed to compute the teaching input, is reduced to a minimum. As soon as one or more objects have been detected in the input sequence, the system always focuses on the largest object. The center of the object is taken as the teaching input.

The paper is structured as follows. In Section 2 we give a brief overview about related research in evolutionary computer vision. Section 3, describes how motion is used to obtain the teaching input. The GPU accelerated evolutionary vision system is described in Section 4. Experiments are presented in Section 5. Conclusions are provided in Section 6.

2 Evolutionary Computer Vision

Evolutionary algorithms can be used to search for a computer vision algorithm when it is not at all clear what such an algorithm should look like. Evolutionary algorithms can also be used to improve upon an existing solution. Work in evolutionary computer vision started in the early 1990s. Lohmann has shown how an Evolution Strategy may be used to find an algorithm which computes the Euler number of an image [15]. Early research focused on evolving low-level operators, e.g. edge detectors [8] or feature detectors [20]. However, evolutionary algorithms were also used for target recognition [12].

In theory, evolutionary methods can be used to evolve adaptive operators which would be optimal or near optimal for a given task [7]. Poli noted very early on, that Genetic Programming [13] would be particularly useful for image processing [19]. Genetic Programming has been used to address a variety of different tasks in computer vision. Johnson et al. have evolved visual routines using Genetic Programming [11]. Current work in evolutionary computer vision ranges from the evolution of low-level detectors [22], to object recognition [14] or even camera calibration [9]. Cagnoni [2] gives a taxonomic tutorial on evolutionary computer vision.

Experiments in evolutionary computer vision usually require enormous computational resources. Each individual of the population has to be evaluated over several generations. That's why experiments in evolutionary computer vision are usually performed off-line. A notable exception (also working with on-line learning) is the work of Mussi and Cagnoni [17]. In our context, multiple alternative image processing algorithms have to be applied to each incoming image. This is only possible through the use of GPU accelerated image processing. Before we describe our evolutionary computer vision system, we first describe how the teaching input is obtained from the input sequence.

3 Fast Detection of Moving Objects in Image Sequences

We have developed a fast method for detecting independently moving objects in image sequences. It is assumed that the camera remains stationary while the image sequence is taken. If the camera itself moves, then information about the ego-motion of the camera could be used to compute a quasi-stationary camera sequence [3].

The method is fast, because image processing operations are reduced to a minimum. Image processing operations are costly because they are applied at least once to every pixel. Operations such as convolution or averaging are particularly costly if they are applied in image space because they require a sliding window and multiple surrounding pixels are accessed for each image pixel.

Thus, we only compute a difference image between two successive images and use this information to update object hypotheses [3]. The approach could be considered to be a minimalistic variant of a particle filtering approach [1,10] where only a single particle is used per object. We compute the average of the differences of the three channels red, green and blue. Differences smaller than than 10% from the maximum (assumed to be noise) are set to zero. We continue by only considering differences larger than this threshold.

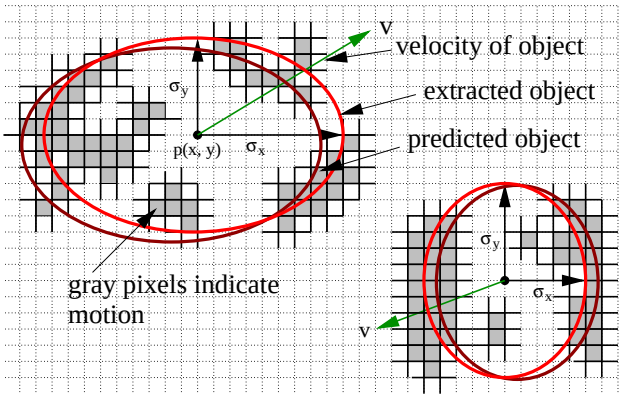

Fig. 1. Differences between two consecutive images are assigned to the nearest object which is predicted from a previous time step. The coordinates of the object predictions (x, y) for the current time step are given by the center of gravity which is computed using all pixels assigned to the object. Two extracted objects are shown.

Let us assume that we have a set of previously extracted objects, i.e. a prediction where objects will be located in the current image. Each object consists of a center of gravity with coordinates $\mathbf{p} = (x, y)$ and also has an associated velocity $\mathbf{v} = (v_x, v_y)$ with which it moves across the image (Figure 1). Each object also has an associated standard deviation in x- and y-direction (σ_x, σ_y). The standard deviations determine the extent of the object. Each pixel with a

difference larger than the threshold contributes to the nearest object. In case a pixel difference cannot be assigned to any object prediction, a new object is created. For newly created object predictions, the center of gravity as well as the standard deviations describing the shape of the object are continuously updated as new pixels are assigned to it.

The center of gravity for the updated object position is computed using all pixels which are located within a distance no more than twice the standard deviation from the center of the object. Pixels further away are assumed to belong to a different object. The standard deviations describing the extent of the object are updated using the same pixels. Let $\mathbf{p}(t_0)$ and $\mathbf{p}(t_1)$ be the positions of the object at time steps t_0 and t_1 respectively. The difference $\mathbf{d} = p(t_1) - p(t_0)$ between the object position for the previous image and the current image is used to update the motion vector of the object. We filter this difference to obtain a smooth approximation of the actual motion vector $\mathbf{v}$ using

$$\mathbf{v}(t_1) = 0.9\mathbf{v}(t_0) + 0.1\mathbf{d}. \tag{1}$$

If an object does not have any associated pixel differences, then the old motion vector is simply added to the center of gravity to predict the new object position for the next time step. If an object does not have any associated pixel differences for three consecutive images, then the object is deleted.

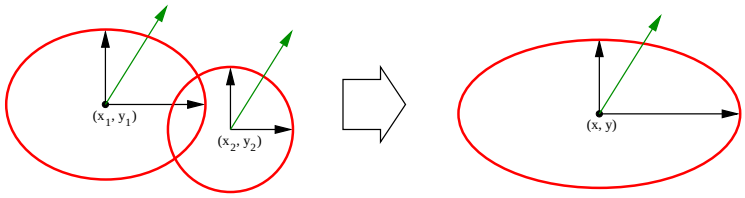

Fig. 2. Nearby objects are merged if they are close to each other.

After existing objects have been updated and new objects have been detected, we iterate over all objects to find objects which need to be merged (Figure 2). Let $\mathbf{p}_1$ and $\mathbf{p}_2$ be the center of gravities of two different objects at the same time step. New objects (for which no motion vector exists yet) are merged if the distance between their center of gravities is smaller than twice the sum of their standard deviations, i.e. if

$$\mathbf{p}_1 - \mathbf{p}_2 \leq 2(\sigma_1 + \sigma_2) \tag{2}$$

with $\sigma_i = \sqrt{\sigma_{x,i}^2 + \sigma_{y,i}^2}$. Existing objects are merged only if the distance between their center of gravities is less than the sum of their standard deviations. They are also merged if the distance is less than twice the sum of their standard deviations provided that they approximately move in the same direction, i.e. their motion vector differs by less than 10%.

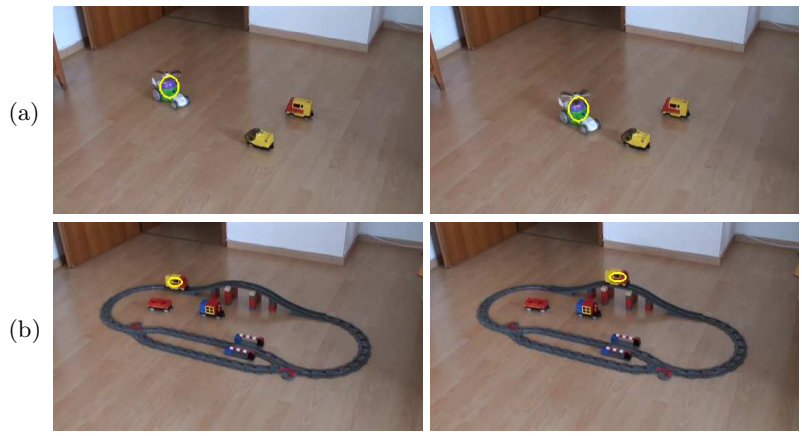

Fig. 3. Moving object detected in two video sequences. The yellow circle marks the detected object. (a) radio-controlled car (b) toy train.

Figure 3 shows how a moving objects are detected and tracked over several frames in two image sequences. We will use the same sequences to evolve a detector for these objects.

4 A GPU Accelerated Evolutionary Vision System

Ebner [6] has developed a GPU accelerated evolutionary vision system. When searching for a solution to a computer vision algorithm, one basically has to assemble computer vision operators in the correct order and also has to decide with which parameters these operators are applied. We are using Cartesian Genetic Programming [16] to automatically search the space of optimal algorithms.

The system works with a $(n_x \times n_y)$ matrix of image processing operators as shown in Figure 4. In addition to this matrix, a set of n_1 high level image processing operators are applied to the input image. We will refer to all of these operators as processing nodes. Each individual of the population codes for an arrangement of image processing operators. High level operators use the original image as input and also create an image as output. Low level operators can have either one or two inputs. Low level operators only perform point operations, i.e. low level operators can be computed by iterating once over all image pixels.

High level operators include operators such as the original image at different scale levels or with a small offset, derivatives in the x- and y-direction, the Laplacian, the gradient magnitude, computation of gray scale images from RGB, segmentation or a convolution. Note that the individuals only have access to a single image frame. We deliberately do not supply two different frames to the system. Individuals should be able to recognize objects in single image frames.

The output of the high level operators is processed by the low level operators inside the $(n_x \times n_y)$ matrix. Data is always fed from left to right. A particu-

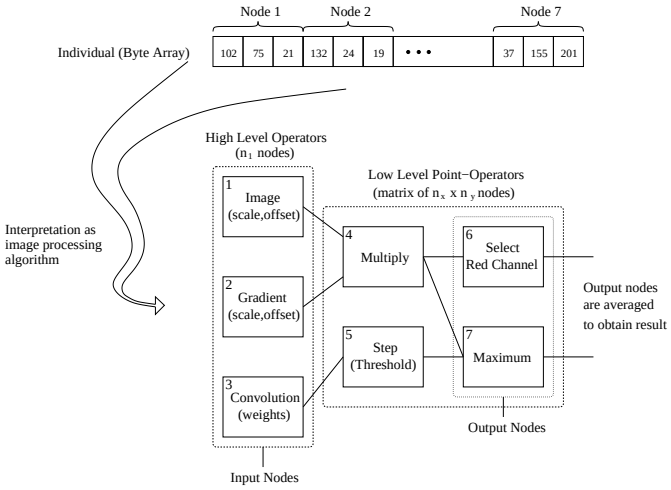

Fig. 4. Sample individual. A linear byte array is mapped to an image processing program consisting of n_1 high level operators and a processing matrix of $n_x \times n_y$ low level, point operators. The output is averaged to obtain the resulting image.

lar node has only access to the data stored in the previous column. The wiring for this matrix is completely under the control of evolution. Low level operators include arithmetic operations, step functions, gate functions, maximum and minimum operations and related functions which access the output from either one or two nodes. The parameters used inside the nodes are also under the control of evolution. A full description of the operators is given by Ebner [6]. The same system is used here except that the constants (0, 0.5 and 1) have been moved from the set of high level operators to the set of low level operators.

Most of the operators are taken directly from the specification of the OpenGL Shading Language (OpenGLSL) [21]. The OpenGL Shading Language was used to accelerate the image processing because it allows easy access to scale spaces which is particularly important for the implementation of the high level operators. It is not clear whether a CUDA [18] implementation would provide any advantage to the GPU acceleration method used here.

Each individual transforms the input image into some other image, the output image. The output image is computed by averaging the output of the n_y rightmost nodes. In order to determine where the object is detected by an individual, we iterate over all image pixels. The position with the maximum output (RGB components are treated as an integer) is taken as the object position. If more than one pixel has the same maximum value, we compute the center of gravity of these pixels. More than 20 pixels having the same maximum value are discouraged by assigning a bad fitness value. We want to the system to clearly mark the detected object in the image.

This representation is referred to as a $n_1 + n_x \times n_y$ representation. It is fully described by Ebner [5,6]. The system works with a linear genotype. Each node

has three associated parameters. The first parameter determines the operator used. The remaining two parameters are either used as parameters for the image processing operator or are used to determine from which previous node the input is received. Each parameter is represented by 8 bits in the genotype.

Each genotype is mapped to the representation shown in Figure 4. A set of n_p individual constitutes the parent population. From the parent population, n_o offspring are generated by applying genetic operators. An additional n_r offspring are generated randomly. Mutation and crossover are used as genetic operators. For each incoming image, parent as well as offspring are evaluated. The best n_p individuals among parents and offspring become the parents for the next input image. When selecting new parents, no double fitness values are allowed. Individuals with the same fitness are assumed to be identical. Using this approach we try to encourage a diverse population of parent individuals.

The fitness of an individual is simply the Euclidian distance between the position detected by the individual and the desired position which is computed using the method described in the previous section.

5 Experiments

For our experiments, we have used $n_p = 3$ parents which generate $n_o = 20$ offspring and $n_r = 20$ randomly generated offspring. Offspring are generated using two point crossover with a crossover probability of $p_{\text{cross}} = 0.5$. The remaining individuals are generated through mutation. The mutation operator either uses a GA-style mutation with a bit wise probability of $p_{\text{mut}} = \frac{2}{l}$ where l is the length of the genotype in bits, or increases or decreases one of the parameters by one. This is to allow also smooth changes of the parameters. A gray code could have been used instead to achieve the same effect.

Given the automated method to extract moving objects, we have objective data to work with and we can rigorously analyze how the method works. This is in contrast to the previous method, where the user had to manually select the object which should be extracted. We work with two video sequences (sample images are shown in Figure 3). The first video sequence shows a radio-controlled car moving around. It consists of 2097 image frames (1m:24s) of size 512×288. The car has a pronounced color which only occurs on the car and not on other objects shown in the sequence. In other words, it is quite easy to come up with an object detector for this car. A simple color filter will do the job. The second video sequence shows a toy train moving around on a track in circles. It consists of 1581 image frames (1m:03s) of size 512×288. The toy train is mostly colored in yellow and red. The same red color can also be found on a wagon which is always present in the image. The yellow color is also shown on the wagon. However, the yellow color on the wagon takes up only a smaller area compared to the yellow on the train.

For our experiments, we turn the evolutionary process on, as long as the desired position differs by more than 25 pixels from the detected position by the individual. The size of the car is approximately 45×50 pixels and the toy

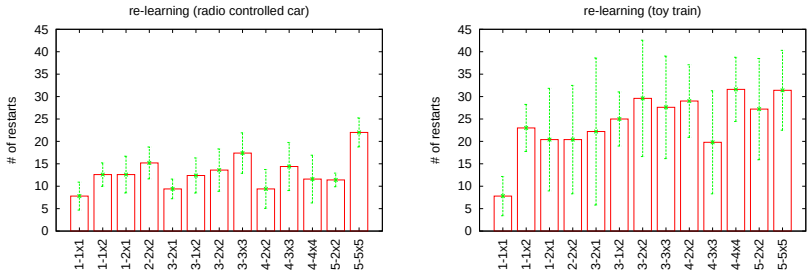

Fig. 5. Average number of restarts for the two image sequences (radio-controlled car and toy train). The standard deviation is also shown.

train is approximately 34×24 pixels. Evolution is turned off if the detected position is very close to the actual position, i.e. the difference between the two is less than 10 pixels for five consecutive frames. If this happens, then only the parent individuals are evaluated and no new offspring are generated. Once the fitness, i.e. the error, rises to more than 25 pixels, then the evolutionary process is turned on again. Note that the individuals do not have to be re-initialized due to the continuous injection of random individuals into the population.

We evaluate how easy or difficult it is to evolve solutions using different $n_1 + n_x \times n_y$ representations. For both image sequences, we measure how often evolution has to be restarted. As described above, evolution has to be restarted if the object is no longer tracked. If evolution has to be restarted only once in a while, then the evolved detectors are very general. If evolution has to be restarted frequently, then the detectors are not general. Such detectors depend on the orientation and/or the size of the object in the image. Figure 5 shows the results for both image sequences. Five experiments were carried out using different random seeds to compute the average.

For the radio controlled car, evolution was only required for 4.5% of the image frames (averaged across all representations and experiments). For 95.5% of the image frames, the object was successfully detected. The object was detected on average with an accuracy of 7 pixels. For the toy train, evolution was only required for 14.6% of the image frames (averaged across all representations and experiments). For 85.4% of the image frames, the object was successfully detected. The object was detected on average with an accuracy of 8 pixels.

It is apparent that the toy train is more difficult to recognize. The data also shows that the problem gets more difficult as the size of the representation is increased. Thus, we want to keep the complexity of the representation minimal while still making sure that the solution is still inside the search space.

Figure 6 shows how long evolution was required to come up with a solution depending on the representation used. Again, the more complex the representation, the longer it took to find good solutions. The toy train sequence is clearly more difficult for the system. For the toy train sequence, it is not sufficient to only use a color detector. The system also has to take the arrangements of the

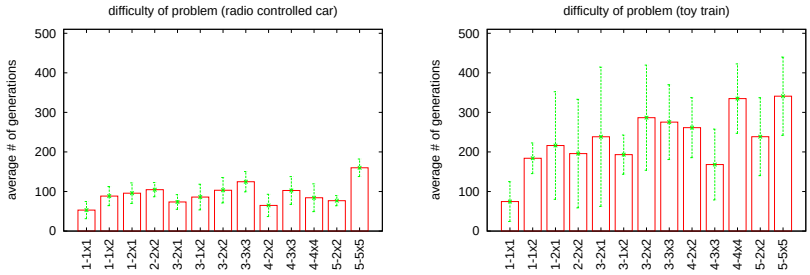

Fig. 6. Number of evolutionary steps. The standard deviation is also shown.

colors into account. To come up with an effective detector for the toy train, the system would have to archive good solutions and to create an overall detector which would recombine the output of several archived detectors. Establishing an archive of detectors will be our next research goal.

6 Conclusions

We have created a GPU accelerated evolutionary image processing system. The system automatically detects moving objects in a video sequence taken with a stationary camera. The coordinates of the detected objects are used to evolve object detectors which are able to recognize the object in a single image. We deliberately use one cue (motion) to train an object detector which is able to recognize objects in images when this cue is not available. The long term goal of this research is to come up with a system which automatically generates object detectors. Our system is a step towards automated learning of object detectors.

References

1. Arulampalam, M.S., Maskell, S., Gordon, N., Clapp, T.: A tutorial on particle filters for online nonlinear/non-gaussian Bayesian tracking. IEEE Trans. on Signal Processing 50(2), 174–188 (2002)
2. Cagnoni, S.: Evolutionary computer vision: a taxonomic tutorial. In: 8th Int. Conf. on Hybrid Int. Systems. pp. 1–6. IEEE Computer Society, Los Alamitos, CA (2008)
3. Ebner, M.: Extraction of moving objects with a moving mobile robot. In: Salichs, M.A., Halme, A. (eds.) 3rd IFAC Symposium on Intelligent Autonomous Vehicles, Vol. II, Madrid, Spain. pp. 749–754. Elsevier Science (1998)
4. Ebner, M.: An adaptive on-line evolutionary visual system. In: Hart, E., Paechter, B., Willies, J. (eds.) Workshop on Pervasive Adaptation, Venice, Italy. pp. 84–89. IEEE (2008)
5. Ebner, M.: Engineering of computer vision algorithms using evolutionary algorithms. In: Blanc-Talon, J., Philips, W., Popescu, D., Scheunders, P. (eds.) Advanced Concepts for Intelligent Vision Systems, Bordeaux, France. pp. 367–378. Springer, Berlin (2009)

6. Ebner, M.: A real-time evolutionary object recognition system. In: Vanneschi, L., Gustafson, S., Moraglio, A., Falco, I.D., Ebner, M. (eds.) Genetic Programming: Proceedings of the 12th European Conference, Tübingen, Germany. pp. 268–279. Springer, Berlin (2009)
7. Ebner, M., Zell, A.: Evolving a task specific image operator. In: Poli, R., Voigt, H.M., Cagnoni, S., Corne, D., Smith, G.D., Fogarty, T.C. (eds.) Joint Proc. of the 1st Europ. Workshops on Evolutionary Image Analysis, Signal Processing and Telecomm., Göteborg, Sweden. pp. 74–89. Springer-Verlag, Berlin (1999)
8. Harris, C., Buxton, B.: Evolving edge detectors with genetic programming. In: Koza, J.R., Goldberg, D.E., Fogel, D.B., Riolo, R.L. (eds.) Genetic Programming, Proceedings of the 1st Annual Conference, Stanford University. pp. 309–314. The MIT Press, Cambridge, MA (1996)
9. Heinemann, P., Streichert, F., Sehnke, F., Zell, A.: 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. pp. 1316–1323. IEEE (2006)
10. Isard, M., Blake, A.: Condensation – Conditional density propagation for visual tracking. Int. Journal of Computer Vision 29(1), 5–28 (1998)
11. Johnson, M.P., Maes, P., Darrell, T.: Evolving visual routines. In: Brooks, R.A., Maes, P. (eds.) Artificial Life IV, Proc. of the 4th Int. Workshop on the Synthesis and Sim. of Living Systems. pp. 198–209. The MIT Press, Cambridge, MA (1994)
12. Katz, A.J., Thrift, P.R.: Generating image filters for target recognition by genetic learning. IEEE Trans. on Pattern Analysis and Machine Int. 16(9), 906–910 (1994)
13. Koza, J.R.: Genetic Programming. On the Programming of Computers by Means of Natural Selection. The MIT Press, Cambridge, MA (1992)
14. Krawiec, K., Bhanu, B.: Visual learning by evolutionary and coevolutionary feature synthesis. IEEE Trans. on Evolutionary Computation 11(5), 635–650 (2007)
15. Lohmann, R.: Bionische Verfahren zur Entwicklung visueller Systeme. Ph.D. thesis, Technische Universität Berlin, Verfahrenstechnik und Energietechnik (1991)
16. Miller, J.F.: An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In: Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference. pp. 1135–1142. Morgan Kaufmann, San Francisco, CA (1999)
17. Mussi, L., Cagnoni, S.: Artificial creatures for object tracking and segmentation. In: Applications of Evolutionary Computing. Proceedings of EvoWorkshops 2008, Naples, Italy, 2008. pp. 255–264. Springer, Berlin (2008)
18. NVIDIA: CUDA. Compute Unified Device Architecture. Version 1.1 (2007)
19. Poli, R.: Genetic programming for image analysis. In: Koza, J.R., Goldberg, D.E., Fogel, D.B., Riolo, R.L. (eds.) Genetic Programming, Proc. of the 1st Annual Conf., Stanford University. pp. 363–368. The MIT Press, Cambridge, MA (1996)
20. Rizki, M.M., Tamburino, L.A., Zmuda, M.A.: Evolving multi-resolution feature-detectors. In: Fogel, D.B., Atmar, W. (eds.) Proc. of the 2nd Am. Conf. on Evolutionary Programming. pp. 108–118. Evolutionary Programming Society (1993)
21. Rost, R.J.: OpenGL Shading Language. Addison-Wesley, Upper Saddle River, NJ, 2nd ed. (2006)
22. Trujillo, L., Olague, G.: Synthesis of interest point detectors through genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference, Seattle, WA. pp. 887–894. ACM (2006)