2012

Real-Time Anomaly Detection in Full Motion Video

Glenn Konowicz,
Old Dominion University

Jiang Li
Old Dominion University

Donnie Self (Ed.)

Original Publication Citation

Konowicz, G., & Li, J. (2012) Real time anomaly detection in full motion video. In D. Self (Ed.), *Full Motion Video (FMV) Workflows and Technologies for Intelligence, Surveillance, and Reconnaissance (ISR) and Situational Awareness, Proceedings of SPIE Vol. 8386* (83860I). SPIE of Bellingham WA. https://doi.org/10.1117/12.919365

Real-time anomaly detection in full motion video

Glenn Konowicz and Jiang Li
Old Dominion University, Norfolk, VA

ABSTRACT

Improvement in sensor technology such as charge-coupled devices (CCD) as well as constant incremental improvements in storage space has enabled the recording and storage of video more prevalent and lower cost than ever before. However, the improvements in the ability to capture and store a wide array of video have required additional manpower to translate these raw data sources into useful information. We propose an algorithm for automatically detecting anomalous movement patterns within full motion video thus reducing the amount of human intervention required to make use of these new data sources. The proposed algorithm tracks all of the objects within a video sequence and attempts to cluster each object's trajectory into a database of existing trajectories. Objects are tracked by first differentiating them from a Gaussian background model and then tracked over subsequent frames based on a combination of size and color. Once an object is tracked over several frames, its trajectory is calculated and compared with other trajectories earlier in the video sequence. Anomalous trajectories are differentiated by their failure to cluster with other well-known movement patterns. Adding the proposed algorithm to an existing surveillance system could increase the likelihood of identifying an anomaly and allow for more efficient collection of intelligence data. Additionally, by operating in real-time, our algorithm allows for the reallocation of sensing equipment to those areas most likely to contain movement that is valuable for situational awareness.

Keywords: Anomaly detection, trajectory clustering, feature tracking, traffic monitoring

1. INTRODUCTION

In a complex video scene with many moving objects there are many challenges involved in identifying and isolating anomalous behavior. This paper present a framework for identifying moving objects within a scene, tracking those objects to identify patterns of behavior, and clustering behaviors to identify those patterns that have not occurred before thus might warrant additional investigation by another system or a human operator. The focus of the paper was on speed of the algorithm and its ability to run in parallel rather than achieving perfect accuracy. This system is designed to run on a camera with a fixed position. However, it does accommodate changes in lighting and shadows as well as changes in motion patterns over time.

Our object tracking algorithm is a simplistic implementation of a template matching algorithm. Similar object tracking techniques that have been investigated before for traffic analysis from a fixed point of view[1], and some of them used more sophisticated algorithms with a moving camera[2]. The use of template matching while also taking into account motion regions was also explored[3]. Other systems used a pool of Kalman models to track objects[4]. A number of algorithms have been studied for anomaly detection in video sequences such as TRACLUS[5] which works by combining trajectories of line segments based on the perpendicular distance, parallel distance, and angle distance from each other, and the use of spectral clustering[6]. The method for anomaly detection proposed in this paper involves matching new trajectories to existing clusters of trajectories and highlighting those trajectories that do no match any existing clusters. We focus on the implementation of a real-time anomaly detection system that can efficiently and reliably detect an anomaly in a video sequence.

2. METHODOLOGY

This system combined portions of several different algorithms in an attempt to minimize the total amount of computation for determining anomalous motion patterns within a given scene.

2.1 Anomaly Detection Framework

The system diagram for anomaly detection is show in Figure 1. Each block was implemented as a C++ class with the basic data passing methods being defined as pure virtual functions. The instantiation of each different block occurs

Full Motion Video (FMV) Workflows and Technologies for Intelligence, Surveillance, and Reconnaissance (ISR) and Situational Awareness, edited by Donnie Self, Proc. of SPIE Vol. 8386, 83860I · © 2012 SPIE · CCC code: 0277-786X/12/$18 · doi: 10.1117/12.919365

through a factory create method. The actual differences in implementation are hidden behind the virtual functions to allow for easily swapping between different algorithms for the same basic functionality. This allows us to compare the accuracy and computational cost between similar algorithms.

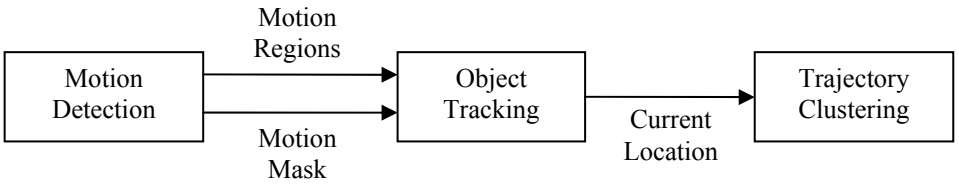


Figure 1. Block diagram of the proposed anomaly detection system.

2.2 Motion detection

Our efficient algorithm for determining which portions of the image contains motion creates a basic background image and then subtracts the background image from the current frame. After that, pixels above a predefined threshold are determined to be in motion. We used a Gaussian average model for modeling the background and it was updated iteratively according to the following equation:

$$B_{t+1} = (1-\alpha)B_t + \alpha I_t \tag{1}$$

where $B_{,t}$ is the background image at time t, I_t is the current frame of a video sequence at time t, and α is the updating constant in between 0 and 1. This model for background modeling is very efficient and uses less memory compared to other algorithms. The Gaussian average filter is best suited for scenes where the background is in slow changing. Though this model is not effective at handling multi-valued background distributions, the disadvantage can be minimized if the errors are relatively small in terms of pixel size[1] and can result in a very fast discernment of the background from foreground. It has been used successfully in other real-time video analysis algorithms[7].

The output of the motion detection is a set of square regions consists of blobs of motion regions. Each motion region is defined as a series of square center x, y, and its width and height. Additionally, the motion detection algorithm outputs a black and white image with all pixels below the threshold being black and all pixels above the threshold being white.

2.3 Object tracking

The proposed system performs object tracking by region matching. Region matching is done in terms of a set of features calculated from the motion mask at the previous locations such as sizes and colors. Our object tracking algorithm consists of two steps. In the first step, it tries to identify the best location in the next video frame for each feature that is already being tracked, and in the second step, it identifies each potential motion region that represents a new feature to be tracked. Once the new locations and sizes for all tracked locations have been determined, we iterate through each potential motion region. Any motion region whose bounding box does not overlap the bounding box of an existing feature already being tracked is added as a new feature to be tracked.

The first step starts by scaling the last few images of the feature to a square image of fixed size. All of the existing images are then combined to form a mean image. This mean image is constructed by averaging the pixel values for each pixel location and excluding those pixels which are not marked as different from the background image. A mask for the feature is then created as the union of all pixels which are different from the background image in any of the previous images. Finally, if the difference between the minimum and maximum value for any color band is above a threshold, it is removed from the mask. Figure 2 shows an example for computing the mean image and its mask. Note the similarity between the shape of the motion mask in 2(b) and the mask computed in 2(g). Also, note the black front of the vehicle in 2(e) and how it causes that portion of the mean image to be excluded in 2(h).

The second step in the tracking is to create a set of possible new locations and sizes. After that, we compute a pixel by pixel difference value between the object region in the current frame and that in each candidate image. All candidate images are normalized by dividing the difference value by the number of pixels, the number of color channels, and the

maximum value for the color depth. Pixels that are not in the motion masks of both the average image and the candidate image to be considered are excluded from the difference computation. The candidates with a difference value closest to zero are then sorted to minimize the change in size. The size and location that minimized the change in size and location from the previous frame is then selected to be the new location and size of the feature.

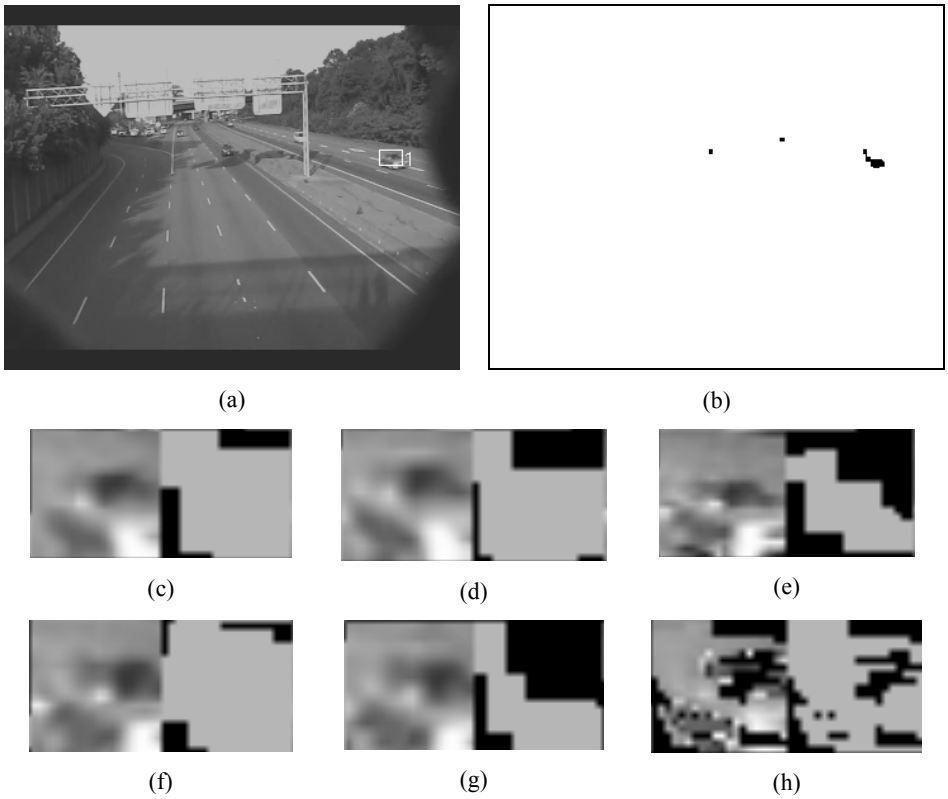


Figure 2. An example of computing the mean image for determining the next location of a feature: (a) The raw input image at time t. (b) The motion mask created by the motion detection algorithm at time t. (c) The feature and its mask at time t_4. (d) The feature and its mask at time t_3. (e) The feature and its mask at time t_2. (f) The feature and its mask at time t_1. (g) The feature and its mask at time t. (h) The mean image and its motion mask.

Once the current sizes and offsets for all of the existing features have been identified, our algorithm proceeds to iterate through the motion regions identified by the motion detection algorithm. Any motion region which falls below a size threshold for width or height is ignored. Any motion region with a bounding box that overlaps the bounding box of an existing region is ignored. The remaining motion regions are then added as new features to the object tracking algorithm.

2.4 Trajectory clustering

The final step in our system is to cluster recorded trajectories into different categories. These trajectories are compared with all existing clusters as demonstrated in a manner similar to other tracking algirthms[8]. Unlike current algorithms for trajectory clustering[9], our algorithm does not split the trajectories up into smaller clusters, but tries to create a large cluster to contain an entire path for a given feature being tracked.

In our clustering algorithm, each trajectory T_i is represented by a list of vectors t_{ij} which are the pixel locations of the object i at time j along the x and y axes of the frame as well as z_{ij} which is the number of frames since the first vector in the scene:

$$T_i = \{t_{i1} \ldots t_{in}\} \text{ where } t_{ij} = \{x_{ij}, y_{ij}, z_{ij}\}$$

By recording the frame number, our clustering algorithm is able to interpolate between frames. This interpolation allows the clustering to be more successful for rapidly moving video sequences where features might jump a large amount of pixels between frames. Clusters are represented as a list of trajectories and their respective time offsets u_{kl} which is the non-negative time difference since the beginning of the cluster k to the first point in trajectory l:

$$C_k = \{c_{k1} \ldots c_{kn}\} \text{ where } c_{kl} = \{T_{kl}, u_{kl}\}$$

The location of the cluster C_k in the scene is computed as the mean distances between all available trajectory values at a given time using the algorithm shown in Figure 3. Because not all trajectories will be the same length and possible noise at the beginnings and ends of the trajectories, a trajectory which defines the beginning of a cluster might not extends to the end of the cluster and the trajectory that defines the end of the cluster might not start at the beginning of the cluster.

- Basic algorithm for determining the cluster location. The detailed steps are as follows
    - Create a new vector v of $\{x_m, y_m, count_m\}$ coordinates and zero all of its values
    - For each vector c_{kl} in C_k
        - For each point t_{ij} in T_{kl}
            - Compute the offset o of the point as $u_{kl}+z_{ij}$
            - Increment the x at offset o in v by x_{ij}
            - Increment the y at offset o in v by y_{ij}
            - Increment the count at offset o in v by 1
    - For each point $\{x_m, y_m, count_m\}$
        - If $count_m$ is non-zero
            - Set x_m to $x_m/count_m$
            - Set y_m to $y_m/count_m$

The determination of whether a trajectory belongs to a given cluster is determined by a single distance threshold *DT*. To start the clustering algorithm, a cluster is initialized as the first trajectory added. As a new trajectory gets added, it is compared with existing clusters to determine the distance from the cluster. The first point for a given trajectory will be compared with all points and offsets in between points for all existing clusters. This will allow the new trajectory to start in the middle or end of an existing cluster. If the distance between the first point of a trajectory and points within existing clusters are not below *DT*, a new cluster is created for the trajectory. As new points are added to the trajectory, each new point is compared with various offsets between the first and the second points within all existing clusters to see if the new point is below *DT* of an existing cluster. If the new point is within *DT*, the cluster for that trajectory is deleted and the trajectory is moved to the matched cluster.

As data points are added to a trajectory the mean Euclidian distance between the location of the cluster with the trajectory removed and the trajectory by itself are compared. If that distance is above *DT*, then the trajectory is removed from the cluster and a new cluster is created. Periodically, the mean Euclidian distance between clusters is compared and if they are below DT they are combined.

As the algorithm progresses, older trajectories are dropped at fixed intervals to keep the total number of trajectories in the clustering algorithm to a manageable size. If all trajectories in a given cluster are deleted then that cluster is deleted. Any time a new cluster is created it is flagged as an anomaly. All of the first few trajectories in a given video will be flagged as anomalies as the clustering algorithm is being trained. Once trained, only those trajectories that do no match a given trajectory will be flagged as anomalies.

3. RESULTS

3.1 Data acquisition

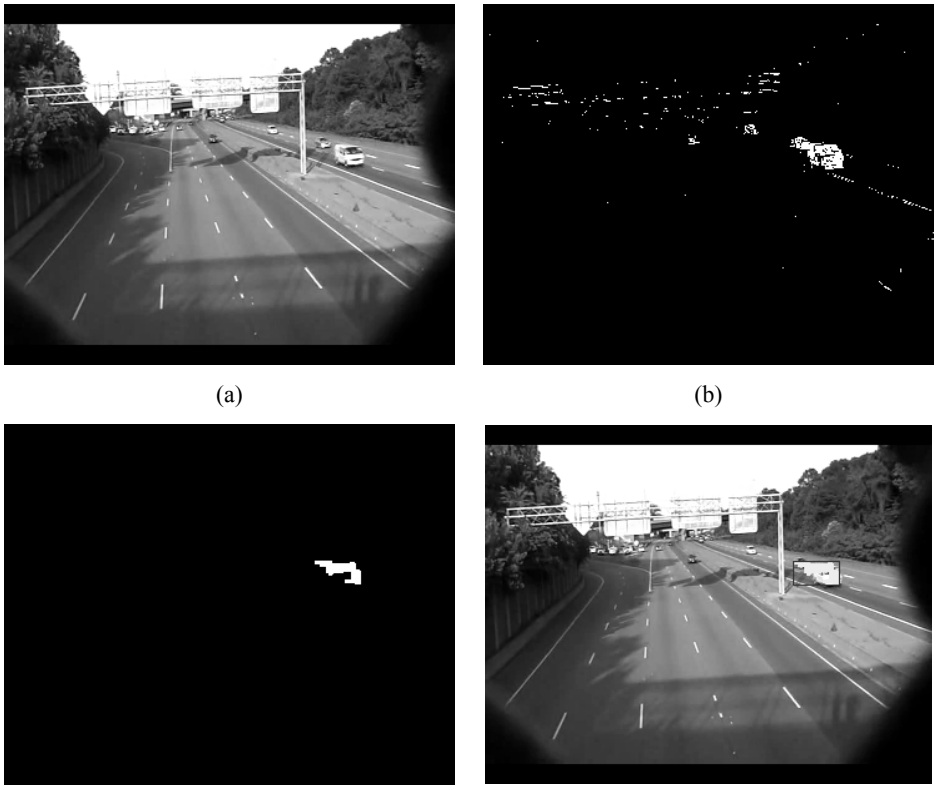
The video data used for testing the proposed system was recorded from a highway overpass where the traffic had a speed limit of 55mph. This video stream allowed the large amount of data in which most of the features follow regular patters as dictated by the traffic laws. Additionally, the speed at which the cars move in and out of the video field of view necessitated a near real-time algorithm which can be processed at a high frame rate. Additionally, the chosen portion of road and time of day forces the tracking algorithm to deal with shadows from signs over the road way, partial occlusion from light poles, and slowly changing background as clouds shift overhead.

3.2 Motion detection

The Gaussian average model used by our algorithm is one of the most efficient techniques for segmenting background from foreground pixels within an image. A Gaussian average model that does not accommodate backgrounds will have difficulties with constantly moving background such as trees in wind. As seen in Figure 4, this leads to much more noise in the background. However, a simple morphological opening and closing operation will remove the noise (Figures 4c and 4d). Once the groups of foreground pixels have been identified, bounding boxes are created for each of the large areas. Since the camera was in a fixed location and most of the noise in any given frame was dispersed enough to be filtered out using a simple opening and closing operation, the extra computation needed to maintain a more sophisticated background model was not necessary.

3.3 Object tracking

By comparing only those images that were detected as in motion, our algorithm allows us to avoid the extra error introduced by comparing a template with new location candidates that also contains background. This helps distinguish features in the scene such as a black car on a black road. In this case, only the parts that are significantly different from the background are used in the identification of the feature and its subsequent future location and size. Given the large amount of flexibility in determining both the location and size, the tracking algorithm was unlikely to select a section of road rather than the vehicle being tracked if the vehicle was approximately the same color as the road. Figure 5 shows the output of a single frame of the object tracking algorithm as well as the internal representation of a single object. Each object currently being tracked is identified by an object number.



(a) (b)

(c) (d)

Figure 3. The motion detection input and output. (a) The raw input image. (b) The difference of the background and the raw input image. (c) The difference image after opening and closing operations. (d) The difference after opening and closing with a bound box.

Figure 4. The output of the feature tracking algorithm. (a) The output from the feature tracking algorithm with a bounding box around the feature and the object number identified

3.4 Trajectory clustering

Since our application requires on-line computation of clusters from various trajectories, we were not able to use traditional off-line clustering algorithms such as a self-organizing map[9]. Instead, we needed to focus on an algorithm that could be updated iteratively. Our approach is similar to other algorithms such as the one describe by Piciarelli [10]. However, our algorithm doesn't try to fit parts of a trajectory into several different clusters. Instead, it attempts to cluster an entire trajectory and even trajectories that might not exactly match in starting and ending locations. The intent of matching an entire trajectory is to recognize those behaviors such as circling a parking lot multiple times as anomaly that is more likely to be identified by using the whole trajectory.

Additionally, the trajectory clustering algorithm tries to interpolate between frames using a basic linear interpolation step. It helps cluster trajectories which might jump many pixels over just a few frames. Furthermore, it gives a measure of resolution to the clustering algorithm. The clustering algorithm compares the distance between trajectories by starting at the first frame and stepping forward by a fraction of a frame. For instance, if the time slice was half of a frame, then the second point of comparison would be the linear interpolation between the first and second frame. If the algorithm needs to run faster, the fraction of a frame between interpolation steps can be increased.

One limitation to the current time based algorithm is that it would not have updates for an object that stopped for a period of time in the middle of its trajectory. For instance, if a car stopped quickly in the middle of the highway then it would just be marked as the end of the trajectory. However, the cars slowing down behind it or driving around the stopped car would be recognized as an anomaly. This limitation provides the advantage of not having continuously updated trajectories for vehicle in a parking lot which are not moving. Only those vehicles that are moving or have recently moved have their trajectories updated.

(a) (b)

Figure 5. One example output of the trajectory clustering algorithm. (a) Early outputs in training with dark lines indicating clusters which have only a few trajectories. (b) Output after several hundred more frames with the light colored lines indicating clusters with multiple trajectories.

3.5 Comparison of computational performance

Since the basic functionality of the tracking algorithm was to take one large image and one small image and then look for an instance of the small image in the large image, we thought OpenCL[11] might be a viable alternative to speed up the processing. However, a basic implementation in OpenCL did not demonstrated sufficient speedup because of the overhead involved in moving the image data as well as the list of candidate locations to the video card and retrieving the esults for each candidate location from the video card. The multi-threaded implementation which just involved a round-robin approach of alternating the possible new locations of a feature on different threads running on a multi-core

processor was able to achieve much better results. Table 1 summarizes the results of the performance analysis on the various implementations of the tracking algorithm.

Table 1. Comparison of CPU times between sequential implementation, multi-threaded implementation, and OpenCL implementation of computing the difference number for 1,800,000 possible combinations of 2x2 feature in a larger image. Normally, only about 50,000 candidate combinations are checked, but a larger number was used in this instance to achieve more consistence timing results.

Implementation	Platform	CPU Time
Single-threaded	AMD Phenom II X6 1055T	3.269s
Single-threaded	Intel Core i7-2760QM	2.020s
Multi-threaded (6 threads)	AMD Phenom II X6 1055T	0.681s
Multi-threaded (8 threads)	Intel Core i7-2760QM	0.557s
OpenCL	AMD Radeon HD6870	16.425s
OpenCL	NVidia Quadro NVS 4200M	1.429s

The overall performance of the entire system was about 20 frames per second. The test machine was a six-core AMD Phenom II system with 16GB of RAM. Each frame of the video feed is 720x576 pixels, but is scaled down to 360x288 at the first phase of processing and remains at that resolution for the rest of the algorithm. The tracking algorithm used six images at 20x20 pixel resolution to represent each feature being tracked.

4. CONCLUSIONS AND FUTURE WORK

The current method of tracking objects in a scene using current multicore processors allows for efficient and effective determination of anomalous motion patterns within a scene. In the performance tradeoff, algorithms that tend to favor speed over accuracy allow for near real-time computation while not significantly decreasing accuracy.

In addition to testing in environments with just one class of object, we intend to expand our anomaly detection algorithm to handle multiple classes of objects. For instance, at an intersection pedestrians would have one set of trajectories and automobiles would have another set of trajectories. These trajectories would need to be clustered independently.

An attempt to minimize the amount of data going between the CPU and the GPU will likely to increase the performance of our OpenCL implementation. This could possibly be accomplished by having the OpenCL kernel compute the list of candidate size and location combinations and only return those combinations which resulted in the lowest difference value thus further reducing the amount of data transferred between the CPU and the video card.

REFERENCES

[1] Koller, D., Weber J., Huang T., Malik J., Ogasawara G., Rao B., and Russel S., "Towards Robust Automatic Traffic Scene Analysis in Real-Time," Proc. ICPR, 126-131 (1994).

[2] Liu, H., Sun, F., "Visual Tracking Using Sparsity Induced Similarity," Proc. ICPR, 1702-1705 (2010).

[3] Lipton, A., Fujiyoshi, H., Patil, R., "Moving target classification and tracking from real-time video," IEEE Workshop on Applications of Computer Vision, 8-14 (1998).

[4] Stauffer, C., Grimson, W., "Learning patterns of activity using real-time tracking," IEEE Transactions on Pattern Analysis and Machine Intelligence 22(8), 747-757, (2000).

[5] Lee, J., Han, J., Whang, K., "Trajectory clustering: a partition-and-group framework," Proc. ACM SIGMOD, 593-604 (2007).

[6] Zhou, Y., Yan, S., Huang, T., "Detecting Anomaly in Videos from Trajectory Simiarity Analysis," IEEE Internation Conference on Multimedia and Expo, 1087-1090 (2007).

[7] Piccardi, M., "Background subtraction techniques: a review," IEEE Conference on Systems, Man and Cybernetics, 3099-3104 (2004).

[8] Dahlbom, A., Niklasson, L., "Trajectory clustering for coastal surveillance," 10th Internation Conference on Information Fusion, 1-8 (2007).

[9] Vesanto, J., Alhoniemi, E., "Clustering of the self-organizing map," IEEE Transactions on Neural Networks 11(3), 586-600 (2000).

[10] Piciarelli, C., Foresti, G.L., Snidaro, L., "Trajectory clustering and its applications for video surveillance," IEEE Conference on Advanced Video and Signal Based Surveillance, 40-45 (2005).

[11] Munshi, A., Gaster, B., Mattson, T., Fung, J., Ginsburg, D., [OpenCL Programming Guide], Pearson Education, Inc., Boston, (2012).