Activity recognition from videos with parallel hypergraph matching on GPUs

Eric Lombardi, Christian Wolf, Oya Celiktutan, Bülent Sankur

Activity recognition from videos with parallel hypergraph matching on GPUs

Eric Lombardi[1] Christian Wolf[1,2] Oya Çeliktutan[3] Bülent Sankur[3]

[1] Université de Lyon, CNRS, LIRIS UMR 5205, France
[2] INSA-Lyon, F-69621, France
[3] Boğaziçi University, Dept. of Electrical-Electronics Eng., Turkey

May 5, 2015

Abstract

In this paper, we propose a method for activity recognition from videos based on sparse local features and hypergraph matching. We benefit from special properties of the temporal domain in the data to derive a sequential and fast graph matching algorithm for GPUs.

Traditionally, graphs and hypergraphs are frequently used to recognize complex and often non-rigid patterns in computer vision, either through graph matching or point-set matching with graphs. Most formulations resort to the minimization of a difficult discrete energy function mixing geometric or structural terms with data attached terms involving appearance features. Traditional methods solve this minimization problem approximately, for instance with spectral techniques.

In this work, instead of solving the problem approximatively, the exact solution for the optimal assignment is calculated in parallel on GPUs. The graphical structure is simplified and regularized, which allows to derive an efficient recursive minimization algorithm. The algorithm distributes subproblems over the calculation units of a GPU, which solves them in parallel, allowing the system to run faster than real-time on medium-end GPUs.

Keywords: Activity recognition Graph matching Parallel algorithms GPU Video analysis

1 Introduction

Many computer vision problems can be formulated as graphs and associated algorithms, since graphs provide a structured and flexible way to inject spatial and structural relationships into matching algorithms. In this paper, it is employed for recognition and localization of actions in videos. The task to detect and localize activities in time and in space and to classify them requires dealing with large quantities of video data in real time.

We formulate the activity recognition task as a correspondence problem between sparse features of short duration model actions and those of longer duration scene videos. The articulated nature of human motions makes it impossible to employ rigid transformations and methods (like RANSAC [14]), while graph matching is able to cope with such non-rigid deformations. The proposed method structures space-time interest points into graphs or hypergraphs using proximity information, as is frequently done in the context of visual recognition (object or activity recognition). The optimal matching between a model video and a scene video is cast as a combinatorial problem to minimize

a function containing terms measuring differences in appearance features as well as terms adressing space-time geometric distortions in the matching.

Despite advances in graph matching due to its popularity and effectiveness, it still remains a challenging task. The computational complexity renders the use of the exact problem on data having a large number of nodes, e.g. video data, intractable in practice. Formulations useful in vision are known to be NP-hard: while the graph isomorphism problem is conjectured to be solvable in polynomial time, it is known that exact subgraph matching is NP-hard [41], and so is subgraph isomorphism [16]. Graph matching solutions in practical problems therefore are of approximate nature.

Recently, computer vision has immensely benefited from development of general purpose computation on graphical processing units (GPUs). Prominent examples are classification in various applications (e.g. pose estimation [37]) and convolution operations, for instance in deep learning architectures [22], feature tracking and matching [38] and patch based image operations, as for instance inpainting [43]. While it is straightforward to profit from parallel architectures if the problem is inherently parallelizable and data-oriented, structured problems are often characterized by complex dependencies which make parallelization of the algorithms difficult. The main inpediments are irregular memory access, tree-based search structures, variable computation and memory units requirement [18].

Whereas most existing work solves the matching problem approximatively, in this work the exact global minimum is calculated, which is made possible by two properties:

- We benefit from two sources, first from the application itself (activity recognition in videos) and the fact that the data are embedded in space time, in particular from specific properties of the time dimension;

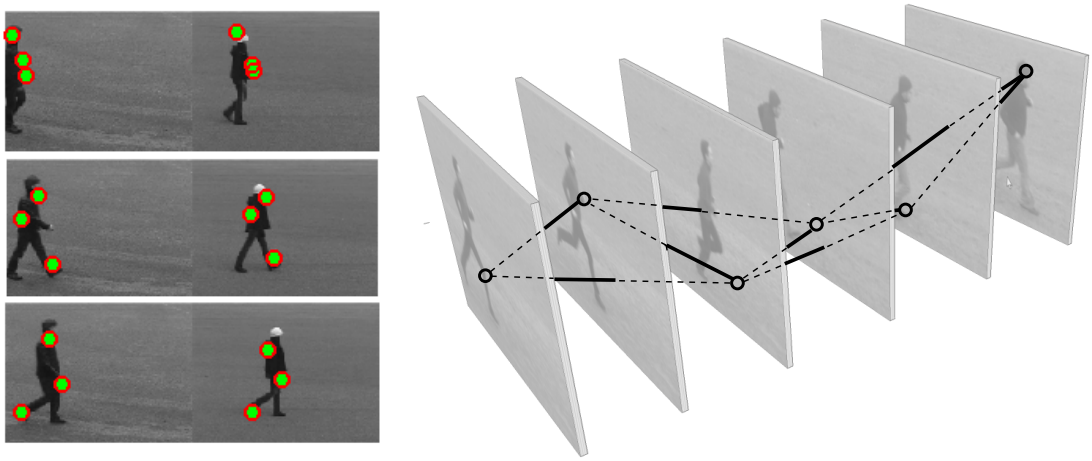
- We approximate the graphical structure and therefore solve a simplified problem exactly and efficiently. The work thus falls into the category of methods calculating the exact solution for an approximated model, unlike methods calculating an approximate solution of an exact problem. In this sense, it can be compared to [7], where the original graph of a 2D object is replaced by a k-tree allowing exact minimization by the junction tree algorithm. Our solution is different in that the graphical structure is not created randomly but is derived from the temporal dimensions of video data.

The linearly ordered nature of actions in time allows matching to proceed through a recursive algorithm over time, which is sequential in nature. However, the subproblems corresponding to calculations associated for individual time instants involve iterations over the domains of discrete variables, which are independent. Faster than real-time performance is achieved through a parallel solution of these subproblems on GPUs.

1.1 Related work

Graphs and graph matching in computer vision — In computer vision, graph matching has been mainly applied for object recognition, but some applications to action recognition problems have recently been reported. In this context, graphs are frequently constructed from sparse primitives like space time interest points [39, 17, 5] or from regions and adjacency information gathered from over-segmented videos [6]. Common matching strategies in this context are off-the-shelf spectral techniques [39] or dynamic time warping on strings of graphs [17].

Other graph related work, albeit not necessarily by matching, is based on stochastic Kronecker graphs [40] for modeling the features pertaining to a specific activity class, chain-graphs [46],



(a) (b)

Figure 1: Videos are represented through space time interest points: (a) examples of successful matches of space time points from the model video (left) to the scene video (right); (b) model videos are structured into graphs with a specific structure using proximity information.

weighted directed graphs [28] where nodes are assigned to frames with salient postures or silhouettes that are shared among different action categories. Here, edges encode the transition probabilities between these postures. Salient postures are used for single-view and multi-view action recognition in [30]. In [30], different actions are linked by chain graphs, called as Action Nets. In [9], the test video is first divided into several subvolumes and the maximum subgraph is searched through branch-and-cut algorithms.

Graph matching and graphs are not the only technique successfully employed for activity recognition. A review of the state of the art of this problem is beyond the scope of this paper, we refer the reader to [1] for a recent survey.

Solving the graph matching problem — Two different formulations dominate the literature on graph matching: (i) Exact matching: a strictly structure-preserving correspondence between the two graphs or at least between their respective parts is searched; (ii) Inexact matching, where compromises in the correspondence are allowed in principle by admitting structural deformations up to some extent. Matching proceeds by minimizing an objective (energy) function.

Most recent papers on graph matching in the computer vision context are based on inexact matching of valued graphs, i.e., graphs with additional geometric and/or appearance information associated with nodes and/or edges. Practical formulations of this problem are known to be NP-hard [41], which makes approximations unavoidable. Two different strategies are frequently used: (i) Calculation of an approximate solution of the minimization problem; (ii) Calculation of the exact solution, most frequently of an approximated model.

Approximate solution — A well known family of methods solve a continuous relaxation of the original combinatorial problem. Zass and Shashua [45] presented a soft hypergraph matching method between sets of features that proceeds through an iterative successive projection algorithm in a probabilistic setting. They extended the Sinkhorn algorithm [20], which is used for soft assignment in combinatorial problems, to obtain a global optimum in the special case when the two graphs have the same number of vertices and an exact matching is desired. They also presented a sampling scheme to handle the combinatorial explosion due to the degree of hypergraphs. Zaslavskiy *et al.* [44] employed a convex-concave programming approach to solve the least-squares problem over the permutation matrices. More explicitly, they proposed two relaxations to the quadratic as-

signment problem over the set of permutation matrices which results in one quadratic convex and one quadratic concave optimization problem. They obtained an approximate solution of the matching problem through a path following algorithm that tracks a path of local minimum by linearly interpolating convex and concave formulations.

A specific form of relaxation is done by spectral methods, which study the similarities between the eigen-structures of the adjacency or Laplacian matrices of the graphs or of the assignment matrices corresponding to the minimization problem formulated in matrix form. In particular, Duchenne *et al.* [12] generalized the spectral matching method from the pairwise graphs presented in [26] to hypergraphs by using a tensor-based algorithm to represent affinity between feature tuples, which is then solved as an eigen-problem on the assignment matrix. More explicitly, they solved the relaxed problem by using a multi-dimensional power iteration method, and obtained a sparse output by taking into account l_1-norm constraints instead of the classical l_2-norm. Leordeanu *et al.* [27] made an improvement on the solution to the integer quadratic programming problem in [12] by introducing a semi-supervised learning approach. In the same vein, Lee *et al.* [25] approached this problem via the random walk concept.

Another approach is to decompose the original discrete matching problem into subproblems, which are then solved with different optimization tools. A case in point, Torresani *et al.* [41] solved the subproblems through graph-cuts, Hungarian algorithm and local search. Lin *et al.* [29] first determined a number of subproblems where each one is characterized by local assignment candidates, i.e., by plausible matches between model and scene local structures. For example, in action recognition domain, these local structures can correspond to human body parts. Then, they built a candidacy graph representation by taking into account these candidates on a layered (hierarchical) structure and formulated the matching problem as a multiple coloring problem. Finally, Duchenne *et al.* [13] extended one dimensional multi-label graph cuts minimization algorithm to images for optimizing the Markov Random Fields (MRFs).

Approximate graphical structure — An alternative approach is to approximate the data model, for instance the graphical structure, as opposed to applying an approximate matching algorithm to the complete data model. One way is to simplify the graph by filtering out the unfruitful portion of the data before matching. For example, a method for object recognition has been proposed by Caetano *et al.* [7], which approximates the model graph by building a k-tree randomly from the spatial interest points of the object. Then, matching was calculated using the classical junction tree algorithm [24] known for solving the inference problem in Bayesian Networks.

A special case is the work by Bergthold et al. [4], who perform object recognition using fully connected graphs of small size (between 5 and 15 nodes). The graphs can be small because the nodes correspond to semantically meaningful parts in an object, for instance landmarks in a face or body parts in human detection. A spanning tree is calculated on the graph, and from this tree a graph is constructed describing the complete state space. The A^* algorithm then searches the shortest path in this graph using a search heuristic. The method is approximative in principle, as hypotheses are discarded due to memory requirements. However, for some of the smaller graphs used in certain applications, the exact solution can be obtained.

Parallel graph matching — Parallel algorithms have been proposed for the graph matching problem for some time. Although both exact solutions and approximate solutions, can be parallized in principle, most of the existing parallel algorithms have been proposed for approximate solution. Many spectral methods, which are approximative, can be naturally ported to a GPU architecture, as the underlying numerical algorithms require matrix operations. Similar matrix operations are employed in [33], where multiple graphs are matched using graduated assignment on GPUs.

Matching two graphs can also be performed by searching the maximum common subgraph of the two graphs [11], a problem which can be transformed (in linear time) to the problem of

searching the maximum clique of a graph. For this problem parallel and GPU algorithms do exist, albeit not very efficient ones [18].

Parallel and GPU algorithms for bi-partite matching have been proposed recently [42, 21]. Bi-partite matching is, however, different and less difficult, as polynomial time algorithms are known for them. These algorithms alternate bidding (proposed assignments) kernels and assignment kernels on the GPU as well as convergence tests on the CPU. A similar problem, unfortunately also called graph matching, matches neighboring vertices of a single graph under unicity constraints. In other terms, a *matching* or independent edge set in a graph is a set of edges without common vertices. GPU algorithms have been proposed for this kind of problem [2].

2 Problem Formulation

Detecting, recognizing and localizing activities in a video stream is cast as a matching problem between a scene video, which can be potentially long or short if the data is processed block-wise in a stream, and a dictionary of relatively much shorter model videos describing the set of activities to recognize. Matching is done by pairs, solving a correspondence problem between the scene video and a single model video at a time. We formulate the problem as a particular case of the general correspondence problem between two point sets with the objective of assigning points from the model set to points in the scene set, such that some geometrical invariance is satisfied. In particular, videos are represented as space-time interest points — see Figure 1a for examples of successful matchings between model point sets and scene point sets. Each point is described by its location in space-time and appearance features, i.e., a descriptor locally representing the space-time region around the point.

The M points of the model are organized as a hypergraph $\mathcal{G}=\{\mathcal{V},\mathcal{E}\}$, where $\mathcal{V}$ is the set of nodes (corresponding to the points) and $\mathcal{E}$ is the set of edges. Let us recall that hypergraphs are a generalization of graphs, where edges, often called *hyperedges*, can link any number of nodes, generally >2. The set of scene nodes is not structured.

Each node i of the model graph is assigned a discrete variable $z_i,\ i = 1..M$, which represents the mapping from the i^{th} model node to some scene node, and can take values in $1..S$, where S is the number of scene nodes. We use the shorthand notation z to denote the whole set of map variables z_i. A solution of the matching problem is given through the values of the z_i, where $z_i{=}j$, $i{=}1..M$, is interpreted as model node i being assigned to scene node $j = 1..S$. Each combination of assignments z evaluates to an energy value in terms of the following energy function $E(z)$:

$$E(z) = \lambda_1 \sum_i U(z_i) \ + \ \lambda_2 \sum_{(i,j,k)\in\mathcal{E}} D(z_i, z_j, z_k) \tag{1}$$

Here, U is a data attached term taking into account the distance between appearance features of point i and its assigned point z_i, D is the geometric distortion between the space-time triangle associated with hyperedge (i, j, k) and the triangle associated with (z_i, z_j, z_k), and λ_1 and λ_2 are weights. For convenience, dependencies on all values over which we do not optimize have been omitted from the notation.

U is defined as the Euclidean distance between the appearance features of assigned points in the case of a candidate match, and it takes a penalty value W^d for dummy assignments which handle situations where a model point is not found in the scene:

$$U(z_i) = \begin{cases} W^d & \text{if } z_i = \epsilon, \\ ||f_i - f'_{z_i}|| & \text{else,} \end{cases} \tag{2}$$

f_i and f'_{z_i} being respectively the feature vector of model point i, and the feature vector of scene point z_i.

The D term is based on angles. Since our data is embedded in space-time, angles include a temporal component not related to scale changes induced by zooming. We therefore split the geometry term D into a temporal distortion term D^t and a spatial geometric distortion term D^g, weighted by a parameter λ_3:

$$D(z_i, z_j, z_k) = D^t(z_i, z_j, z_k) + \lambda_3 D^g(z_i, z_j, z_k) \tag{3}$$

where the temporal distortion D^t is defined as time differences over two pairs of nodes of the triangle:

$$D^t(z_i, z_j, z_k) = \Delta(i, j) + \Delta(j, k) \tag{4}$$

with:

$$\Delta(i, j) = |(t(i) - t(j)) - (t'(z_i) - t'(z_j))| \tag{5}$$

Here, $\Delta(i, j)$ is the time distortion due to the assignment of model node pair (i, j) to scene node pair (z_i, z_j). The temporal distortion term penalizes the discrepancy in the extent of time between model node pairs and the corresponding scene node pairs. The model node pairs should not be too close or too far from each other likewise the scene node pairs. Finally, D^g is defined over differences of angles:

$$D^g(z_i, z_j, z_k) = \left|\left| \begin{array}{l} a(i, j, k) - a'(z_i, z_j, z_k) \\ a(j, i, k) - a'(z_j, z_i, z_k) \end{array} \right|\right|. \tag{6}$$

Here, $a(i, j, k)$ and $a'(z_i, z_j, z_k)$ denote the angles subtended at point j for, respectively, model triangle indexed by (i, j, k) and scene triangle indexed by (z_i, z_j, z_k). $\|.\|$ is the L2 norm. The difference between angles takes into account the circular domain of angles.

2.1 Approximations

In the context of activity recognition, the geometric data are embedded in space-time. We make the following assumptions relative to the temporal domain to derive an efficient minimization algorithm:

Assumption 1: Causality — While objects (and humans) can undergo arbitrary geometrical transformations like translation and rotation, which is subsumed by geometrical matching invariance in our formulation, human actions can normally *not* be reversed. In a correct match, the temporal order of the points should be retained, which can be formalized as follows

$$\forall\, i, j : t(i) \leq t(j) \Leftrightarrow t'(z_i) \leq t'(z_j) \tag{7}$$

where the notation $t(i)$ stands for the temporal coordinate of model interest point i, and $t'(z_i)$ stands for the temporal coordinate of scene interest point z_i.

Assumption 2: Temporal closeness — Another reasonable assumption is that the extent of time warping between model and scene time axes must be limited. In other words, two points which are close in time must be close in both the model set and the scene set. Since our graph is created from proximity information (time distances have been thresholded to construct the hyperedges), this can be formalized as follows:

$$\forall\, i, j, k \in \mathcal{E} : |t'(z_i) - t'(z_j)| < T \ \wedge\ |t'(z_j) - t'(z_k)| < T \tag{8}$$

where T is a parameter.

We have shown in [8], that this problem can be solved in polynomial time if the data are embedded in space-time, as opposed to the general class of problems, which is NP-hard [41].

Due to the sequential nature of activities, graphs obtained from data embedded in space-time are generally elongated in time, i.e. the graphical structure is heavily influenced by the temporal order of the nodes. This is particularly true in the case of graphs constructed from space time interest points, which are commonly very sparse. Typical interest point detectors extract only few such points per frame — typically between 0 and 5 [23]. We take advantage of this observation to restrict the model graph by keeping only a single interest point per model frame. This is done by choosing the most salient one, i.e., the one with the highest confidence of the interest point detector.

We also restrict the set $\mathcal{E}$ of model graph edges to connections of each model point i to its two immediate frame-wise predecessors $i-1$ and $i-2$ as well as to its two immediate successors $i+1$ and $i+2$. This creates a planar graph with triangular structure, as illustrated in Figure 1b. A video is described by the way this planar graph twists in space time, as well as the appearance features associated with each node. According to the visual content of a video, there may be frames which do not contain any space time interest points, and therefore no nodes in the model graph. These empty frames are not taken into account, e.g. in Equation (8), when triplets of consecutive frame numbers are considered.

2.2 Minimization

The neighborhood system of this simplified graph can be described in a very simple way using the node indices of the graph, similar to the dependency graph of a second order Markov chain. The general energy given in (1) can therefore be expressed simpler as follows:

$$E(z) = \sum_{i=1}^{M} U(z_i) + \sum_{i=3}^{M} D(z_i, z_{i-1}, z_{i-2}) \tag{9}$$

where we also have absorbed λ_1 and λ_2 into U and D, respectively. The elongated form of the graph allows us to derive an efficient inference algorithm for calculating $\widehat{z} = \arg\min_z E(z)$ based on the following recursion:

$$\alpha_i(z_{i-1}, z_{i-2}) = \min_{z_i} \Big[U(z_i) + D(z_i, z_{i-1}, z_{i-2}) + \ \alpha_{i+1}(z_i, z_{i-1}) \Big] \tag{10}$$

with the initialization

$$\alpha_M(z_{M-1}, z_{M-2}) = \min_{z_M} \left[U(z_M) + D(z_M, z_{M-1}, z_{M-2}) \right] \tag{11}$$

During the calculation of the trellis, the arguments of the minima in Equation (10) are stored in a table $\beta_i(z_{i-1}, z_{i-2})$. Once the trellis is completed, the optimal assignment can be calculated through classical backtracking:

$$\widehat{z_i} = \beta_i(z_{i-1}, z_{i-2}), \tag{12}$$

starting from an initial search for z_1 and z_2:

$$(\widehat{z_1}, \widehat{z_2}) = \arg\min_{z_1, z_2} [U(z_1) + U(z_2) + \alpha_3(z_2, z_1)] \tag{13}$$

The computational complexity and the memory complexity of the algorithm are defined by the trellis, a $M \times S \times S$ matrix, where each cell corresponds to a possible value of a given variable. The

calculation of each cell requires to iterate over all S possible combinations of z_i. It is easy to see that the computational complexity is $O(S^3M)$ and that the memory complexity is $O(S^2M)$.

Exploiting the different assumptions on the spatio-temporal data given above, the computational complexity can be decreased further. A large amount of combinations from the trellis can be pruned applying the following constraints:

- given variable z_i, all values for its predecessors z_{i-1} and z_{i-2} must be necessarily *before* z_i, i.e. lower.
- given variable z_i, we will allow a maximum number of T possibilities for the values of the successors z_{i+1}, z_{i+2}, which are required to be *close*.

These pruning measures decrease the complexity to $O(SMT^2)$, where T is a small constant measured in the number of frames. Since T is a small constant, the computational complexity therefore becomes linear on the number of points in the scene: $O(SM)$.

Let us note that no such prior pruning is applied to the scene frames, which therefore may contain an arbitrary number of points. At a first glimpse it could be suspected that the single-point-per-frame approach could be too limited to adequately capture the essence of an action sequence. Experiments have shown, however, that the single chain performs surprisingly well. It should be noted again, that no restrictions have been imposed on the scene, in other words, none of the scene points have been eliminated.

3 A parallel solver for GPUs

Solving the matching problem requires computing Equations (11), (10), (12) and (13). However, the computational and memory complexity are dominated by the requirement to solve (10) for different indices i and for different combinations of z_{i-1} and z_{i-2}, which boils down to filling a three dimensional array with values, taking into account certain dependencies between these values. In the following section we will present a parallel solver for this problem designed for modern GPUs. Although the system has been implemented using the hardware independent OpenCL library, we kept the description as library independent as possible. For this reason, and to make the paper self-contained, we first define some common terms (and refer to Figure 2 for a block scheme of modern GPU architecture):

Kernel — a kernel is a piece of code which is executed by the GPU in parallel on several independent hardware units;

Work-item — the execution of a kernel on a single data package is called a work-item;

Work-group — work-items can be grouped into work-groups; all work-items of one work-group share data in local memory;

Global memory — global GPU memory can be accessed by all work-items. It is slower than local GPU memory;

Local memory — local GPU memory is shared by the work-items of the same work-group. It is faster than global GPU memory;

RAM — the computer's (classical) main memory used by the CPU. It cannot be directly accessed by work-items, thus code running on the GPU.

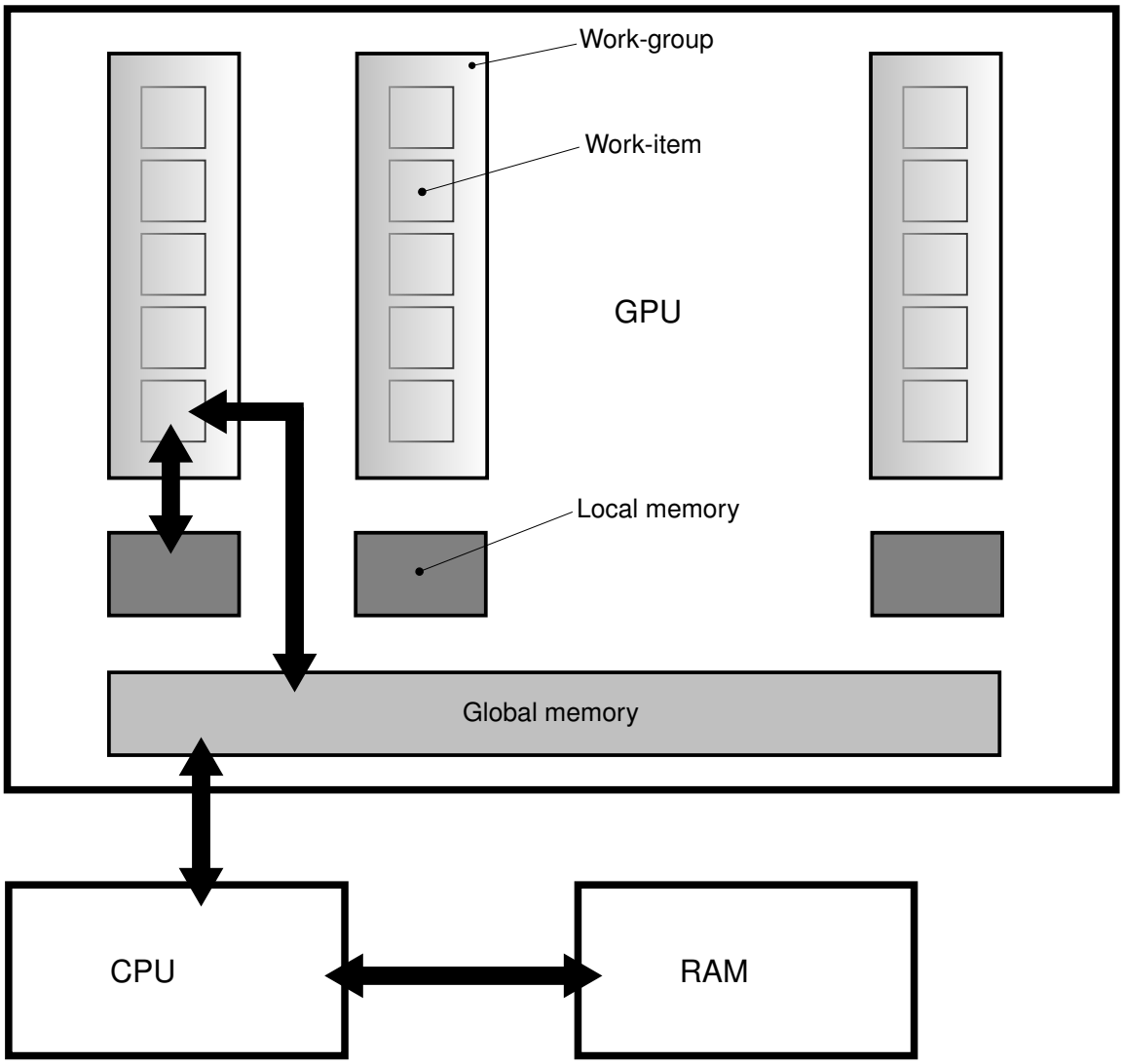


Figure 2: The architecture of a CPU+GPU system as seen by the hardware independent OpenCL library.

Given the restrictions of memory access and execution on the GPU, parallel algorithms follow a classical three-step process:

- transfer data from RAM to global GPU memory;
- eventually transfer data from global GPU memory to different local GPU memory banks;
- execute the kernel multiple times in parallel on the GPU;
- transfer results from GPU memory to RAM

Dependencies between results may require more complex schemes and/or multiple iterations of this process.

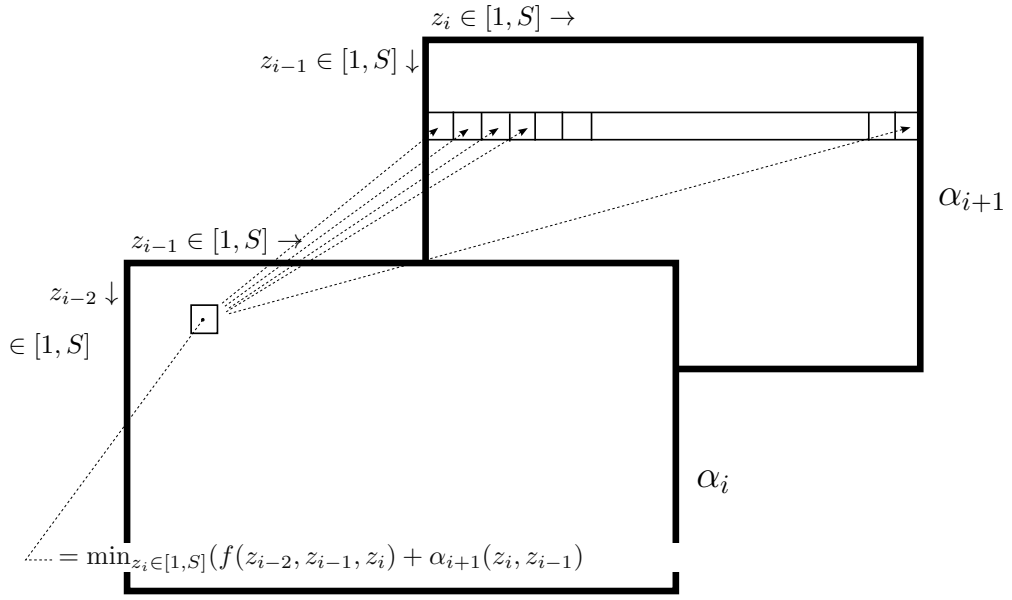


Figure 3: A visualization of the 3D trellis of Equation (10) as a series of 2D tables of size $S \times S$. Each cell in table α_i is a result of a minimization operation taking as input results from a row in table α_{i+1}.

3.1 Defining the GPU kernel

The recursion on i in Equation (10) produces (and works on) a 3D structure with 3 coordinates : i, z_{i-1}, z_{i-2}. Here, i is the model frame/node index, which can also be interpreted as a temporal coordinate, and which takes values in $\{1..M\}$; z_{i-1} and z_{i-2} are assignment variables which can each take values in $\{1..S\}$. It is convenient to visualize this trellis as a series of 2D tables, as illustrated in Figure 3. The kernel code will execute the minimization in Equation (10), where a single work-item deals with a triplet of parameters given by i, z_{i-1} and z_{i-2}. Accordingly, a total number of MS^2 kernel executions (work-items) is required to fill the trellis.

The dependencies in Equation (10) make it impossible to execute all kernels in parallel: it is easy to see, that a full row in α_{i+1} is required as input for each cell in α_i. A scheduling is needed, which executes kernels in parallel with a block-wise sequential ordering, insuring that a whole α_{i+1} row is available, before the corresponding cell in α_i is executed. This scheduling could theoretically be done by the kernel itself through synchronization points. However, kernel-wise synchronization is only possible for work-items of the same work-group. The amount of work-items in the studied problem (MS^2) makes it unreasonable to use a single work-group for the whole problem. Indeed, convenient values are M=30 and S=60. This corresponds to 1 seconds of model video and 2 seconds of scene video at 30 frames per second, using one node per frame. The amount of work items involved in the matching of the model graph to the scene graph, MS^2, is then 108000. This value is far above the capacity of even a high end GPU like the GTX580, where a workgoup is limited to 1024 work items. If we match a model against larger blocks of the scene, for instance entire videos, then this value will be even higher (see also table 2 for comparisons of two different scenarios with S=754 and S=60).

A different solution is to perform synchronization through the CPU, i.e. launch parallel executions of a single array α_{i+1} on the GPU with the CPU taking over control between iterations over i. In other words, at the end of the kernel execution, the fall back to CPU execution acts as a syn-

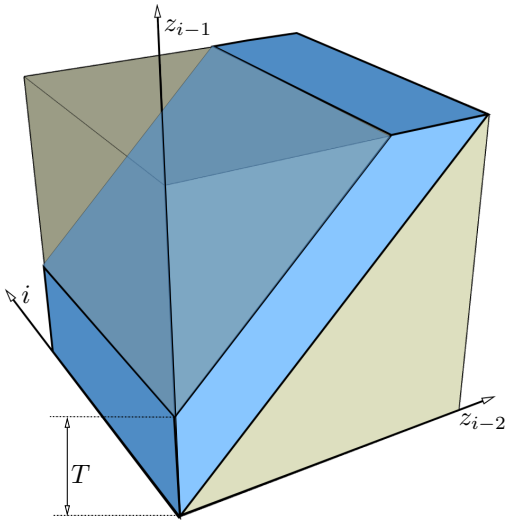


Figure 4: The 3D trellis of size $M \times S^2$ calculated by the recursive minimization algorithm. The shaded cross-section of size $\sim M \times S \times T$ in the middle corresponds to the admissible combinations of i, z_{i-1} and z_{i-2}.

chronisation point for all work-items. Note, that the resulting scheduling is not different; a slight performance loss is due to the control flow change between the GPU and the CPU. This leads to a two-dimensional kernel, the dimensions being z_{i-1} and z_{i-2}. A single loop in the kernel iterates over z_i to perform the minimization operation in (10).

3.2 Pruning the trellis

The causality assumption (7) and the temporal closeness assumption (8) restrict the admissible combinations of z_i, z_{i-1} and z_{i-2}, and therefore allow us to prune combinations in the trellis. In particular, for a given value of z_{i-1}, the value of z_{i-2} is restricted to the interval $]z_{i-1} - T, z_{i-1}[$. This limits the admissible values of the trellis to a cross section of size $\sim S \times T$ in the 3D volume, as illustrated in Figure 4. The validity tests for these constraints are precomputed and stored in a boolean array. A similar reasoning restricts the values of z_i, c.f section 3.4.

3.3 Precomputing the unary terms

The complexity of the unoptimized kernel is dominated by the calculation of the unary terms $U(z_i)$, which are calculated in each iteration of the minimization loop, and which correspond to the Euclidean distance between the feature vector of a model node and the feature vector of a scene node. The size F of the appearance feature vector may vary, common sizes are 50 to 200 components (162 in the case of our HoG/HoF features).

Assumptions (7) and (8) decrease the part of U for a work-item from $O(FS)$, to $O(FT)$, where $T \leq S$ (see also sub section 3.4). The unoptimized computational complexity for a whole array α_i is $O(FST^2)$ and for the whole trellis it is $O(FSMT^2)$. Another way to derive the same result is to consider that each minimization in equation (10) takes place over T iterations, that the minimization is performed SMT times (c.f. the blue cross section in figure 3), and that one term is of $O(F)$.

The unary terms take as input the model node and scene node. Therefore, out of the SMT^2 calls to U, only SM different input arguments exist, which can be pre-calculated on the GPU by a separate kernel. The pre-computed unary terms are stored in a look-up table of size $S \times M$ in

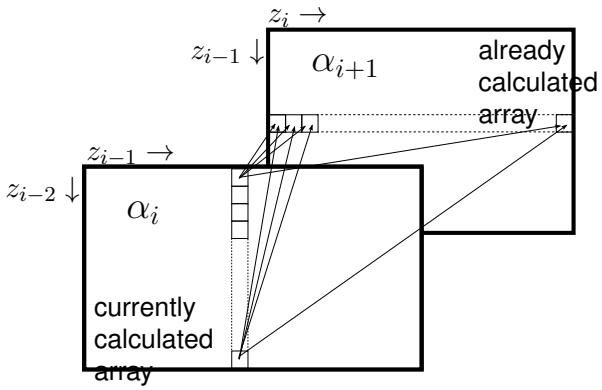


Figure 5: Dependencies: one column of array α_i is handled by the work-items of the same work-group. The values of this column depend on a single row of previous array α_{i+1}.

the GPU global memory. They are later used by the main GPU kernel, which calculates one α_i array. When the α_i kernel evaluates the matching of model node i to scene node z_i, it reads the corresponding pre-computed value of U in the look-up table.

Precomputing U for all combinations of model nodes and scene nodes saves T^2 calls to U. The pre-computations themselves can be done in $O(FSM)$, so total computational complexity is reduced from $O(FSMT^2)$ to $O(SMT^2 + FSM)$. For typical values of T=10 and F=162, the speed up factor is ~ 61.

3.4 Precomputing the minimization loop boundaries

The minimization loop involved in equation (10) is executed inside each kernel. During one kernel execution, i.e. for one work-item, z_{i-1} and z_{i-2} have a fixed value. The loop is then executed over z_i, whose value are limited by z_{i-1} and z_{i-2}, according to causal constraint (7) and temporal closeness constraint (8). Applying the two constraints above and performing some algebraic simplifications results in two equations involving z_i:

$$\begin{cases} t'(z_i) \geq t'(z_i - 1) \\ t'(z_i) < t'(z_i - 2) + T \end{cases} \tag{14}$$

Equations (14) define an interval in temporal coordinates, which needs to be translated into an interval in scene nodes. Let us recall that the minimization in equation (10) is over possible scene nodes. As opposed to model graphs, scenes can feature multiple nodes per frame, as illustrated in figure (6). This distribution of nodes over frames can be pre-calculated and stored in a table:

$$minnode(f) = \inf(\{n : t'(n) = f\}), n = 1...S$$

where $minnode(f)$ gives the first node of a given scene frame f, assuming that the scene nodes are sorted in temporal (i.e. frame) order. Then, the boundaries of the minimization loop in equation (14) can be directly derived as follows:

$$\begin{cases} \min(z_i) = minnode(t'(z_{i-1})) \\ \\ \max(z_i) = minnode(t'(z_{i-2}) + T) - 1 \end{cases}$$

where we took advantage of the fact that the maximum node of a frame f is equal to $minnode(f + 1) - 1$.

Function $minnode(.)$ only depends on the distribution of the nodes in the scene. It can be pre-computed for each scene block, stored in GPU memory, and then shared by all work-items.

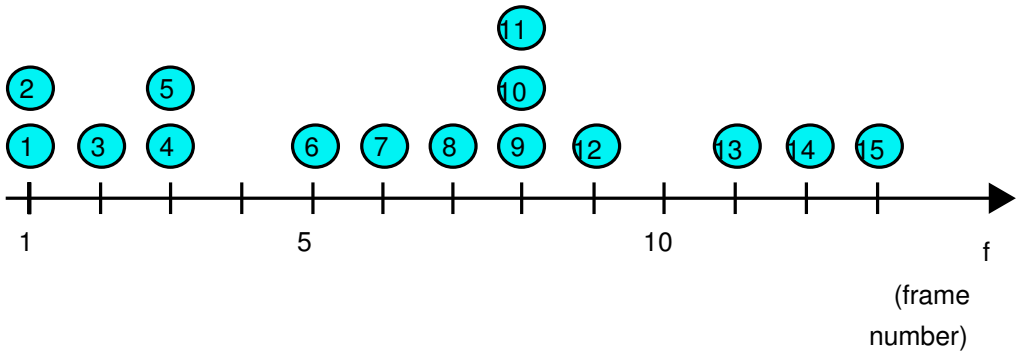


Figure 6: Example of a distribution of scene nodes over scene frames: as opposed to models, scenes may feature multiple nodes per frame. For example, frame 8 has nodes 9, 10 and 11.

Algorithm 1: The kernel executed for each work-item.

Kernel

Copy a cell of α_{i+1} from global mem. to local mem.;
Wait for synchronization between all work-items;
Compute one cell of array α_i (minimization in (10));

end

3.5 Work-groups and local GPU memory

Local GPU memory (NVidia "shared memory" or OpenCL "local memory") offers faster access than global GPU memory. Local memory is associated to a work-group, i.e., only work-items of this work-group can access it. Read and write operations to local memory can be synchronized between work-items of a single work-group. The goal is to optimize performance by performing a single transfer from global memory to local memory, followed by several accesses to local memory directly done by the kernels executions. Efficiency is guaranteed by organizing work-groups such that work-items can share common data.

As illustrated in Figure 5, all cells of a single column of an array α_i depend on the same row of array α_{i+1}. This can easily be derived from the recursive Equation (10): calculating one column $\alpha_i(z_{i-1}, z_{i-2})$ for fixed i and z_{i-1} requires the values $\alpha_{i+1}(z_i, z_{i-1})$ for fixed $i+1$ and z_{i-1} over varying z_i, which corresponds to a row in the precedently calculated array. This naturally leads to a configuration where one column of the currently calculated array α_i is organized in a single work-group, where the column corresponds to the expression $\alpha_i(z_{i-1}, z_{i-2})$ for fixed i and z_{i-1}.

The memory transfer of a whole row for a single workgroup is distributed over its work-items. As all tables α_i are of square shape (the number of columns equals the number of rows), a single value is transferred by each work-item at the beginning of the kernel execution. After a synchronisation point, the minimization routine itself is executed. This results in the kernel structure given in Algorithm 1.

As a result, S^2 read operations from slow global memory per work-group have been replaced by S read operations from global memory plus S^2 read operations from fast local memory. The total number of operations slightly grows by a small factor of $(S + S^2)/S^2$, which in practice is smaller than 1.017 for the configuration we employed (S=60). However, the total access time decreases due to faster memory accesses.

Algorithm 2: The CPU side algorithm.

```
Copy full data from RAM to global GPU memory;
for i=M to 1 do
    In parallel do
        Compute array α_i on GPU (run kernel);
        if i<M then
            Copy array α_{i+1} from GPU to RAM;
        end
    end
end
Copy array α_1 from GPU to RAM;
```

3.6 GPU memory persistence

All model and scene data must be transfered from the CPU controlled RAM to the GPU, and hardware constraints force a transfer to global GPU memory. Let us recall that after a parallel computation of an array α_i, execution falls back to the CPU, which controls the iterations over the different arrays α_i for different indices i. This could potentially involve a large number of memory transfers between CPU RAM and global GPU memory, as data needs to be exchanged between the CPU and the GPU before and after each iteration.

Data is persistent in global GPU memory between kernel executions, which can be exploited to minimize the number of memory transfers. The full model data and the currently processed scene block are initially entirely transferred to the global GPU memory, and stay persistent between iterations over the tables α_i. The α_i arrays also stay in GPU memory between successive kernel executions.

3.7 Parallelizing calculations and transfer

After calculation of the trellis, the optimal point assignment is calculated with the backtracking step given in Equation (12). This step is of very low complexity (a single loop over M iterations) and inherently sequential, therefore it is performed on the CPU. The whole trellis (tables $\alpha_i(z_{i-1}, z_{i-2})$ for $i \in [1, M]$) must be transferred from global GPU memory to the CPU accessible RAM. Modern GPU architectures provide the ability to parallelize kernel executions and data transfer, which is here exploited to copy the results available in array α_{i+1} while kernel executions are computing the entries of α_i. The CPU side of the matching algorithm is given in Algorithm 2:

3.8 Memory structure optimizations

An important concept in efficient computing on most GPU architectures is to ensure coalesced accesses to global GPU memory. Practically, this means that data in the global GPU memory has to be organised in contiguous blocks of multiples of 128 bytes to maximize access rate inside work-items. As a whole row of table α_{i+1} needs to be read for each cell (work-item) in α_i, we store the different tables α_i in row order.

3.9 Computational complexity and run-time

As mentioned in section 2.2, taking advantage of all approximations results in a computational complexity of $O(SMT^2)$, where T is a small constant (T=10 in our experiments). However, that

Work	Average perf.	Run-time	Evaluation	Remarks on run-time
Ta *et al.* [39]	91.2%	1.86s	LOOCV	s/frame, matching with 98 model graphs
Borzeshi *et al.* [5]	70.2%	N/A	Split 8/8/9	
Brendel & Todorovic [6]	N/A	10s	N/A	Matching 1000 nodes graph with 2000+ nodes graph
Lv & Nevatia [30]	N/A	5.1s	N/A	s/frame
Savarese *et al.* [35]	86.8%	N/A	LOOCV	
Ryoo & Aggarwal [34]	93.8%	N/A	LOOCV	
Mikolajczyk & Uemura [31]	95.3%	5.5s to 8s	LOOCV	s/frame (-5s if SVM are not used)
Baccouche *et al.* [3]	95.8	N/A	LOOCV	N/A
Jiang *et al.* [19]	93.4%	N/A	LOOCV	
Our method on CPU	91.0%	0.2s	Split 8/8/9	s/frame, matching 754-nodes scene with 50 model graphs
Our method on GPU	91.0%	0.02s	Split 8/8/9	s/frame, matching 754-nodes scene with 50 model graphs
Our method on GPU	91.0%	0.0035s	Split 8/8/9	s/frame, matching 60-nodes scene with 50 model graphs

Table 1: Comparison with the state-of-the-art methods on the KTH database (LOOCV = Leave-one-out-cross validation).

does not necessarily mean that run-time is a function of $O(SMT^2)$, as parallel processing is not yet taken into account. Actual run-time is of course considerably lower. If we consider Q work-units (cores), run-time is of $O(\lceil \frac{ST}{Q} \rceil MT)$: M kernel calls are launched, each one performing a minimization over T iterations, and each kernel call scheduling ST work-items. In the case where enough work-units are available on the GPU to perform all ST work-items of one call in parallel, i.e. $Q \geq ST$, run-time is actually of $O(MT)$.

4 Experimental Results

The presented matching algorithm has been evaluated on a real-world application, namely action recognition in video sequences. The widely used public KTH database [36] has been chosen as test dataset. It contains 25 subjects performing 6 actions (*walking*, *jogging*, *running*, *handwaving*, *handclapping* and *boxing*) recorded in four different scenarios including indoor/outdoor scenes and different camera viewpoints, totally 599 video sequences (one is corrupted). The subdivision of the sequences is the same as the one provided on the official dataset website[1]. This results in 2391 subsequences in total.

The images of the KTH database are of size 160x120. However, only the preprocessing steps depend on the image size, the complexity of the matching algorithm itself is independent of it.

We use spatio-temporal interest points extracted with the 3D Harris detector [23] to constitute the nodes of the graph. Appearance features are the well-known HoG-HoF features extracted with the publicly available code in [23]. The weighting parameters are set so that each distortion measure

[1] http://www.nada.kth.se/cvap/actions/

Implementation	#Scene nodes	#Scene frames	Time/single model Tot(ms)	per fr(ms)	All 50 models Time/fr (ms)	
CPU: Intel Core 2 Duo, E8600 @ 3.33 Ghz, Matlab/C(mex)	754	723	2900	4.01	200.5	
Nvidia GeForce GTS450, 192 cuda cores @ 1566 MHz, mem bandwidth 21.3 GB/sec	754 60	723 55	748 4	1.03 0.07	51.5 3.5	 real time!
Nvidia GeForce GTX560, 336 cuda cores @ 1660 MHz, mem bandwidth 128 GB/sec	754 60	723 55	405 4	0.56 0.07	28 3.5	real time! real time!
Nvidia GeForce GTX580, 512 cuda cores @ 1544 MHz, mem bandwidth 192 GB/sec	754 60	723 55	**178** 4	0.25 0.07	21 3.5	real time! real time!

Table 2: Running times in milliseconds for two different GPUs and for two different scene block sizes. The last column on the right gives times per frame for matching the whole set of 50 model graphs. The bold blue value of **178 ms** is comparable to the values in table 3.

has the same range of values: λ_1=0.6, λ_2=0.2, λ_3=5, T=10 (λ_3 is explained in the appendix).

All experiments use the leave-one-subject-out (LOSO) strategy. We augmented the number of model graph prototypes by selecting different sub-sequences and constructed each graph consisting of 20 to 30 frames containing at least one or more salient interest points. Action classes on the unseen subjects are recognized with a nearest prototype classifier (NPC). The distance between model and prototypes is based on the matching energy given in Equation (1). However, experiments showed that best performance is obtained if only the appearance terms $U(.)$ are used for distance calculation instead of the full energy (1).

We learned a set of discriminative prototypes using sequential floating forward search (SFFS) [32]: model graph prototypes are created from the training subjects and the prototype selection is optimized over the validation set.

The maximum admissible size of each work-group depends on the GPU itself and on the kernel specifically implemented for the problem. In our case, and for the Nvidia GeForce GTX 560 GPU, the limit is 768 work-items per work-group, which in practise is higher than the number of work-items we need: S=60 if scene blocks of 2 seconds are matched in video streaming mode, and S=754 if whole scene videos of 30 seconds are matched.

Floating point operations on Nvidia GPUs use different rounding algorithms than the FPUs on Intel CPUs, which may results in slightly different values obtained in the trellis. However, this did not impact the recognition results.

We would like to point out that many results have been published on the KTH database, but most of the results cannot be compared due to different evaluation protocols, as has been studied on the detailed report on the KTH database in [15]. For completeness, we compare our performance with state-of-the-art methods. In Table 1, we report average action recognition performance and computational time for the aforementioned methods in Section 1. The results have been copied from the original papers. Although run-time calculations and used protocols differ between the papers, this table gives an overall idea that our proposed method is extremely competitive in terms

Implementation	Optimizations	Time (ms)
CPU: Intel Core 2 Duo, E8600 @ 3.33 Ghz, Matlab/C(mex)	3.2, 3.3	2900
Basic GPU Kernel Nvidia GeForce GTX580	3.2, 3.3	794
GPU Kernel with all alg. optimizations Nvidia GeForce GTX580	3.2, 3.3 + 3.4	326
GPU Kernel with all alg. and archit. optimizations Nvidia GeForce GTX580	3.2, 3.3, 3.4 + 3.5, 3.6, 3.7, 3.8	**178**
GPU Kernel with all alg. and archit. optimizations Nvidia GeForce GTX580	3.2, 3.3, 3.4 + 3.5, 3.6, 3.7, 3.8 (No closeness constraint: $T{=}\infty$)	1853

Table 3: Contribution of different optimizations on execution time. Matching one model graph of 30 nodes to one scene graph of 754 nodes and 723 frames.

Table 4: Comparison of different model types with and without restriction to a single point per frame (M: number of model nodes, S: number of scene nodes, $\overline{M}$: number of model frames, $\overline{S}$: number of scene frames, R: maximum number of interest points per frame in the *scene* sequence, $N = 3$: number of matched single-chain-single-point model).

Method	Complexity	Accuracy
Proposed method (1 point per frame)	$O(SMT^2)$	91.0%
N=3 points per frame, method 1 (greedy)	$O(\overline{SM}T^2R^N)$	90.2%
N=3 points per frame, method 2 (independent chains)	$O(SMT^2N)$	90.8%

of run-time with at the same time providing recognition performance comparable to the state of art. The best performing methods require several seconds of calculation per frame, whereas our method requires only 0.0035 seconds per frame. Our GPU algorithm is faster in several orders of magnitude.

The GPU implementation allows faster than real-time performance on standard medium end GPUs, e.g. a Nvidia GeForce GTS450. Table 2 compares run-times of the CPU implementation in Matlab/C (the critical sections were implemented in C) and the GPU implementation running on different GPUs with different characteristics, especially the number of calculation units. The run-times are given for matching a single model graph with 30 nodes against scene blocks of different lengths. If the scene video is cut into smaller blocks of 60 frames, which is necessary for continuous video processing, real time performance can be achieved even on the low end GPU model. With these smaller chunks of scene data, matching all 50 graph models to a block of 60 frames (roughly 2 seconds of video) takes roughly 3 ms regardless of the GPU model.

We can observe that the results for the three different GPUs (GTS450, GTX560 and GTX580)

are identical when models are matched against scenes of 60 nodes. In this case, the number of cuda cores is higher than the amount of requested work items per call, i.e. per frame. Therefore, adding additional cuda cores does not give us an increase in performance. Other architectural differences (frequency, bandwidth) between the video cards are negligible and beyond the precision of time measurement. If we match one model against a whole video (754 frames), which requires much more work items, then the extra amount of cuda cores of the GTX560 and GTX580 cards makes a large difference in performance.

The processing time of 3 ms per frame is very much lower than the limit for real time processing, which is 40 ms for video acquired at 25 frames per second. Additional processing will be required in order to treat overlapping blocks, which increases running time to 6 ms per frame. The times given above also do not include interest point detection and feature extraction, but fast implementations of these steps do exist. As mentioned, in our experiments we used Laptev et al's STIP points [23], which performed slightly better than other detectors. Its pure CPU implementation requires 82 ms per frame. However, our own implementation of Dollar et al.'s points [10] runs in 2 ms per frame including the detection of HoG features (also on CPU). These run-times can be decreased porting the algorithms to GPU, especially Dollar et al.s detector, which essentially proceeds through linear filtering in time and space. The convolution operations can be easily performed on GPUs.

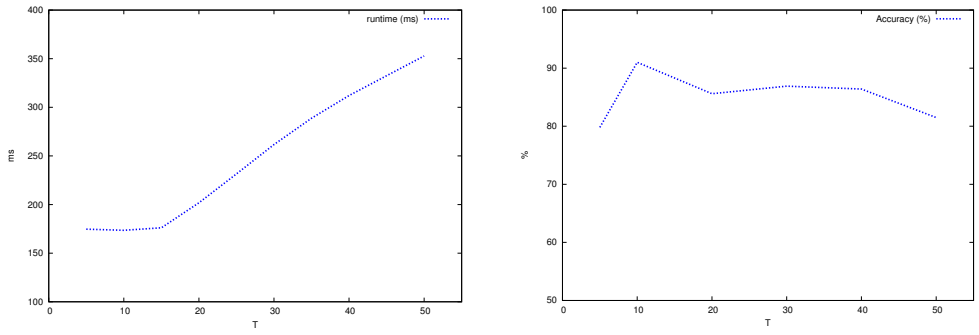
In table 3, we compare execution times between the CPU implementation of the matching algorithm, and several implementations on GPU, with different types of optimizations. Comparing the CPU version running time (2900 ms) with the non-optimized GPU version (794 ms), we can see that the parallel hardware architecture of the GPU is almost 4 times faster than the CPU. The algorithmically optimized version of the GPU kernel (including optimizations described in sections 3.2, 3.3, 3.4) takes 326 ms to run, and is then 2.4 times faster than the non-optimized version. Finally, the GPU architectural optimizations described in sections 3.5, 3.6, 3.7, 3.8 bring an additional speedup of factor 2 (178 ms vs. 326 ms).

The last line in the same table (table 3) shows a comparison with the fully optimized GPU version without temporal closeness constraint (1853 ms), giving the influence of the restriction of temporal warping. Figure 7 shows this effect of parameter T (restricting temporal warp) on performance. The parameter retained for all other experiments is T=10, and at this configuration we find the value of 178 ms also given in tables 2 and 3. Loosening the restrictions will not only significantly increase computational complexity, it will also decrease recognition performance. The restriction due to the parameter T allows to remove wrong assignments. No restrictions ($T = +\infty$) will lead to a runtime of 1853 ms.

We also performed experiments to check the influence of the restriction to a single interest point per frame. The original formulation given in equation (1) is of polynomial complexity, but still too complex to solved in real time. We compared the proposed approximation to two different models allowing several interest points per frame :

Multiple points 1: a greedy approximation of the the full model, where interest points assignments are decoupled from frame assignments. First, for each model frame and for all possible assigned scene frames, the optimal interest point assignments are made based on unary terms U and inter-frame deformation terms D. Then, frames are assigned using the proposed model.

Multiple points 2: creation of several single point models (several second order chains), each of which is solved independently. The resulting matching distance is given as the average over the different chains.



(a) (b)

Figure 7: Performance as a function of parameter T: (a) run-time; (b) classification performance (accuracy). No restrictions ($T = +\infty$) will lead to a runtime of 1853 ms (Nvidia GTX580, 754 scene nodes).

More details on these models are given in [8]. Results on the KTH are given in table 4. Results of the multiple point methods are comparable, even slightly lower. After investigating the results in detail, we believe that taking into account more points hurts performance because this decreases the invariance properties of the method. Space-time points are naturally very sparse, and taking more points actually leads to including less stable and less robust points.

5 Conclusion

An efficient parallel algorithm for activity recognition has been presented, which resorts to parallel hypergraph matching on GPUs. The traditional problem of irregular computation flow in graph matching has been addressed by restricting the graph to a regular structure, which allows efficient inference by a recursive function working on a 3D trellis. The values in the trellis are computed in a block-wise parallel manner. The method is competitive with the state of the art in terms of recognition performance while at the same time being faster in several orders of magnitude. The current implementation is faster than real time on medium-end GPUs even though only a part of the computing units are currently used. Further speed-up could be gained by matching a scene video block against several model graphs in parallel and by distributing the cells of the multiple trellis over the computing units of the GPU.

Acknowledgement

This work has been partially funded by the ANR project SoLStiCe (ANR-13-BS02-0002-01), a project of the grant program “ANR blanc”.

References

[1] Aggarwal, J.K., Ryoo, M.S.: Human activity analysis: a review. ACM Computing Surveys (2011)

[2] Auer, B.F., Bisselig, R.H.: A gpu algorithm for greedy graph matching. In: Facing the Multicore Challenge II - Lecture Notes in Computer Science, pp. 108–119 (2012)

[3] Baccouche, M., Mamalet, F., Wolf, C., Garcia, C., Baskurt, A.: Spatio-temporal convolutional sparse auto-encoder for sequence classification. In: British Machine Vision Conference (BMVC) (2012)

[4] Bergtholdt, M., Kappes, J., Schmidt, S., Schnörr, C.: A Study of Parts-Based Object Class Detection Using Complete Graphs. International Journal of Computer Vision **87**(1-2), 93–117 (2010)

[5] Borzeshi, E.Z., Piccardi, M., Xu, R.Y.D.: A discriminative prototype selection approach for graph embedding in human action recognition. In: ICCVW (2011)

[6] Brendel, W., Todorovic, S.: Learning spatiotemporal graphs of human activities. In: ICCV (2011)

[7] Caetano, T., Caelli, T., Schuurmans, D., Barone, D.: Graphical models and point pattern matching. IEEE Tr. on PAMI **28**(10), 1646–1663 (2006)

[8] Celiktutan, O., Wolf, C., Sankur, B., Lombardi, E.: Fast exact hyper-graph matching for spatio-temporal data. Journal of Mathematical Imaging and Vision (to appear) (2014)

[9] Chen, C., Grauman, K.: Efficient activity detection with max-subgraph search. In: CVPR (2012)

[10] Dollár, P., Rabaud, V., Cottrell, G., Belongie, S.: Behavior recognition via sparse spatio-temporal features. In: ICCV VS-PETS. Beijing, China (2005)

[11] Donatello, C., Foggia, P., Vento, M.: Challenging complexity of maximum common subgraph detection algorithms: A performance analysis of three algorithms on a wide database of graphs. Journal of Graph Algorithms and Applications **11**(1), 99–143 (2007)

[12] Duchenne, O., Bach, F.R., Kweon, I.S., Ponce, J.: A tensor-based algorithm for high-order graph matching. In: CVPR, pp. 1980–1987 (2009)

[13] Duchenne, O., Joulin, A., Ponce, J.: A graph-matching kernel for object categorization. In: ICCV (2011)

[14] Fischler, M., Bolles, R.: Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Communications of ACM **24**(6), 381–395 (1981)

[15] Gao, Z., Chen, M., Hauptmann, A., Cai, A.: Comparing evaluation protocols on the kth dataset. In: Human Behavior Understanding, vol. LNCS 6219, pp. 88–100 (2010)

[16] Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman (1979)

[17] Gaur, U., Zhu, Y., Song, B., Roy-Chowdhury, A.: A string of feature graphs model for recognition of complex activities in natural videos. In: ICCV (2011)

[18] Jenkins, J., Arkatkar, I., Owens, J., Choudhary, A., Samatova, N.: Lessons learned from exploring the backtracking paradigm on the gpu. In: International conference on Parallel processing, pp. 425–437 (2011)

[19] Jiang, Z., Lin, Z., Davis, L.S.: Recognizing human actions by learning and matching shape-motion prototype trees. PAMI (2012)

[20] Knight, P.A.: The sinkhorn-knopp algorithm: Convergence and applications. SIAM Journal on Matrix Analysis and Applications **30**(1), 261–275 (2008)

[21] Kollias, G., Sathe, M., Schenk, O., Grama, A.: Fast parallel algorithms for graph similarity and matching. Tech. Rep. CSD-TR-12-010, Purdue University (2012)

[22] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Neural Information Processing Systems (NIPS) (2012)

[23] Laptev, I., Marszalek, M., Schmid, C., Rozenfeld, B.: Learning realistic human actions from movies. In: CVPR, pp. 1–8 (2008)

[24] Lauritzen, S., Spiegelhalter, D.: Local computations with probabilities on graphical structures and their application to expert systems. Journal of the Royal Statistical Society B **50**, 157–224 (1988)

[25] Lee, J., Cho, M., Lee, K.: Hyper-graph matching via reweighted random walks. In: CVPR X (2011)

[26] Leordeanu, M., Hebert, M.: A spectral technique for correspondence problems using pairwise constraints. In: ICCV, pp. 1482–1489. Washington, DC, USA (2005)

[27] Leordeanu, M., Zanfir, A., Sminchisescu, C.: Semi-supervised learning and optimization for hypergraph matching. In: ICCV 2011 (2011)

[28] Li, W., Zhang, Z., Liu, Z.: Expandable data-driven graphical modeling of human actions based on salient postures. IEEE Tr. CSVT (2008)

[29] Lin, L., Zeng, K., Liu, X., Zhu, S.C.: Layered graph matching by composite cluster sampling with collaborative and competitive interactions. CVPR **0**, 1351–1358 (2009)

[30] Lv, F., Nevatia, R.: Single view human action recognition using key pose matching and viterbi path searching. In: CVPR (2007)

[31] Mikolajczyk, K., Uemura, H.: Action recognition with appearance motion features and fast search trees. CVIU **115**(3), 426–438 (2011)

[32] Pudil, P., Ferri, F.J., Novovicov, J., Kittler, J.: Floating search methods for feature selection with nonmonotonic criterion functions. In: ICPR, pp. 279–283 (1994)

[33] Rodenas, D., Serratosa, F., Sol-Ribalta, A.: Parallel graduated assignment algorithm for multiple graph matching based on a common labelling. In: Graph-Based Representations in Pattern Recognition, *Lecture Notes in Computer Science*, vol. 6658, pp. 132–141 (2011)

[34] Ryoo, M.S., Aggarwal, J.K.: Spatio-temporal relationship match: video structure comparison for recognition of complex human activities. In: ICCV (2009)

[35] Savarese, S., Delpozo, A., Niebles, J., Fei-Fei, L.: Spatial-temporal correlatons for unsupervised action classification. In: WMVC. Los Alamitos, CA (2008)

[36] Schuldt, C., Laptev, I., Caputo, B.: Recognizing human actions: a local svm approach. In: ICPR, pp. 32–36 (2004)

[37] Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Blake., A.: Real-time human pose recognition in parts from single depth images. In: CVPR. Colorado Springs, USA (2011)

[38] Sinha, N., Frahm, J.M., Pollefeys, M., Genc, Y.: Gpu-based video feature tracking and matching. In: Workshop on Edge Computing Using New Commodity Architectures, vol. 278 (2006)

[39] Ta, A.P., Wolf, C., Lavoue, G., Başkurt, A.: Recognizing and localizing individual activities through graph matching. In: AVSS (2010)

[40] Todorovic, S.: Human activities as stochastic kronecker graphs. In: ECCV (2012)

[41] Torresani, L., Kolmogorov, V., Rother, C.: Feature correspondence via graph matching: Models and global optimization. In: ECCV, pp. 596–609 (2008)

[42] Vasconcelos, C., Rosenhahn, B.: Bipartite graph matching computation on gpu. In: International Conference on Energy Minimization Methods in Computer Vision and Pattern Recognition, pp. 42–55 (2009)

[43] Wang, G., Xiong, Y., Yun, J., Cavallaro, J.: Accelerating computer vision algorithms using opencl framework on the mobile gpu-a case study. In: ICASSP, pp. 2629—2633 (2013)

[44] Zaslavskiy, M., Bach, F., Vert, J.: A path following algorithm for the graph matching problem. IEEE Tr. on PAMI **31**(12), 2227–2242 (2009)

[45] Zass, R., Shashua, A.: Probabilistic graph and hypergraph matching. In: CVPR (2008)

[46] Zhang, L., Zeng, Z., Ji, Q.: Probabilistic image modeling with an extended chain graph for human activity recognition and image segmentation. ITIP (2011)