Towards Model Checking Video Streams Using VoxLogicA on GPUs

Laura Bussi[1,2(✉)] , Vincenzo Ciancia[2] , Fabio Gadducci[1] ,
Diego Latella[2] , and Mieke Massink[2]

[1] Dipartimento di Informatica, Università di Pisa, Pisa, Italy
`laura.bussi@phd.unipi.it, fabio.gadducci@unipi.it`
[2] Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", Consiglio Nazionale delle Ricerche, Rome, Italy
`{l.bussi,v.ciancia,d.latella,m.massink}@isti.cnr.it`

Abstract. We present a feasibility study on the use of spatial logic model checking for real-time analysis of high-resolution video streams with the tool `VoxLogicA`. `VoxLogicA` is a voxel-based image analyser based on the Spatial Logic for Closure Spaces, a logic catered to deal with properties of spatial structures such as topological spaces, graphs and polyhedra. The underlying language includes operators to model proximity and reachability. We demonstrate, via the analysis of a series of video frames from a well-known video game, that it is possible to analyse high-resolution videos in real-time by exploiting the speed-up of `VoxLogicA-GPU`, a recently developed GPU-based version of the tool, which is 1–2 orders of magnitude faster than its previous iteration. Potential applications of real-time video analysis include medical imaging applications such as ultrasound exams, and other video-based diagnostic techniques. More broadly speaking, this work can be the first step towards novel information retrieval methods suitable to find information in a declarative way, in possibly large collections of video streams.

1 Introduction

The topological approach to Spatial Model Checking, introduced in [8,9], provides tools and techniques, typical of the Formal Methods community, for the analysis of graph-based spatial data. Some early, prominent applications of the technique can be found in the area of Smart Cities and Smart Transportation (see [12,13], and also the more recent work in [15]). The spatial model checking approach of `VoxLogicA`[1] (see [4] and the tutorial [11]) aims at encoding Expert Knowledge in executable form in the domain of Medical Imaging. The focus is

---

[1] `VoxLogicA` is Free and Open Source Software. Source code and binaries are available at https://github.com/vincenzoml/VoxLogicA.

on procedures that are intelligible by domain experts and not solely by programmers. By design, this idea can operate in conjunction with other forms of analysis; for instance, a `VoxLogicA` procedure could be used to delimit a larger area within which a Machine Learning procedure can be used to provide a more detailed analysis. This would give rise to a form of Hybrid Artificial Intelligence. Such larger areas could reflect specific domain oriented guidelines or analysis protocols, for example to make sure that the Machine Learning procedure focuses on the right region of interest to reduce the number of false positives. The spatial model checker `VoxLogicA` automatically computes the result of `ImgQL` (Image Query Language) queries on (possibly large) image datasets.

In [4], a ten-lines-long `ImgQL` specification was used for the segmentation of Glioblastoma, a common form of brain tumour, in circa 200 cases from the 2017 "Brain Tumour Segmentation (BraTS) challenge" dataset. Spatial Model Checking is fast, and specifications are intelligible to domain experts and can be discussed in the wider community for further improvement. In terms of accuracy, the procedure scores among the top ranking methods of BraTS 2017 – the state of the art in the field, currently dominated by machine-learning methods – and it is comparable in quality to manual delineation by human experts (see [4] for further details comparing our results with the 18 alternative techniques used in BraTS 2017 that were applied to at least 100 cases of the dataset). The segmentation procedure takes only a few seconds per case to complete.

In [3], `VoxLogicA` has been used for skin lesions segmentation, which is the first task in melanoma diagnosis. An `ImgQL` procedure was applied to images of skin lesions from two datasets released by ISIC (International Skin Imaging Collaboration) for the 2016 challenge – a training set (900 images) and a test set (379 images) – obtaining results, in terms of accuracy and computational efficiency, in line with the state of the art.

Graphical Processing Units (GPUs) are high-performance, massively parallel computational devices that are available in various sizes (and computing power), in diverse machines ranging from smart phones, tablets, laptops, workstations to large-scale cloud-based computing facilities. GPU computing differs from the *multi-core* paradigm of modern CPUs in many respects: the execution model is *Single Instruction Multiple Data*; the number of computation cores is high; the memory model is highly localised and synchronisation among parallel threads is very expensive. In [7] a GPU-based version of `VoxLogicA` was introduced. The obtained speedup on synthetic benchmarks, using an office workstation equipped with a good consumer GPU, was between one and two orders of magnitude. The main challenges that were faced in [7] concerned the minimisation of the expensive read/write operations from and to the GPU memory, and turning each algorithm into a massively parallel one. To date, `VoxLogicA-GPU` implements the core logical primitives of `VoxLogicA` on GPU, including reachability (based on connected component labelling). Such effort shares some motivation with a recent trend on implementing formal methods on GPU [5,16,17,19,20].

The work in [3,4] concerns the analysis of individual images. In our future work, we are interested in extending the technique to the analysis of medical

imaging applications, such as ultrasound exams, and other video-based diagnostic techniques in real time. Such analysis would require processing of videos in real time. This paper presents a case study that aims at investigating whether spatial model checking can be used to analyse high-resolution videos in real-time. As we shall see, the GPU implementation is the key to achieve this. Since `VoxLogicA` cannot yet load videos directly, our experiments operate on individual video frames, saved on disk as separate `png` images, but these frames are loaded in a single model-checking session in batches (say, each one corresponding to 5 s of video) in order to maximise throughput. Loading single frames introduces a high overhead due to loading and saving separate files. Therefore in order to measure the "real" execution time, we defined a strategy to mitigate the impact of such overhead, which will not be present e.g., if frames are streamed from a webcam. After this, we demonstrate that the GPU implementation operates in real-time with large margins for future improvement.

As an abstract, but still feature-rich, example, we use a video of the Pac-Man video game[2]. The results show that with the spatial model checking approach we are able to precisely identify the video frames where interesting aspects of gameplay are present.

The paper is organised as follows. Section 2 provides the relevant background on spatial model checking. Section 3 presents the experimental set-up and Sect. 4 presents the results. Section 5 concludes the work and provides an outlook on further research.

2   Background: Spatial Model Checking on GPU

We briefly review the syntax of the Spatial Logic for Closure Spaces (SLCS), defined in [8,9], and its interpretation, restricted to the case of two-dimensional images which is currently handled by `VoxLogicA-GPU`. For the general definition on so-called *closure spaces*, and the link between (multi-dimensional) images, graphs, closure spaces and topological spatial logics we refer the reader to [1,4,9]. The syntax of the logic we use in this paper is its most up-to-date rendition, where the *surrounded* connective from [9] is a derived one, whereas reachability is primitive, as in [2,10,14]. Given set P of *atomic propositions*, with $p \in P$, the syntax of the logic is described by the following grammar:

$$\phi ::= p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \mathcal{N}\phi \mid \rho\,\phi_1[\phi_2] \tag{1}$$

The logic is interpreted on the pixels of an image $\mathcal{M}$ of fixed dimensions. The truth values of a formula ϕ on *all* the pixels can be rendered as a binary image of the same dimensions of $\mathcal{M}$. Therefore, in particular, **atomic propositions** correspond to binary images. However, concretely, when working on images, atomic propositions also include constraints (e.g., thresholds) on imaging features, such as intensity, or red, green, blue colour components. **Boolean operators** are defined pixel-wise: $\neg\phi$ is the complement of the binary image representing ϕ,

---

[2] PAC-MAN$^{\text{TM}}$ & ©1980 BANDAI NAMCO Entertainment Inc.

and $\phi_1 \wedge \phi_2$ is binary image intersection. The **modal** formula $\mathcal{N}\phi$ is satisfied by a pixel x that shares a vertex or an edge with any of the pixels that satisfy ϕ (adopting the so-called *Moore neighbourhood*); in imaging terminology, this is the *dilation* of the binary image corresponding to ϕ. The **reachability** formula $\rho \, \phi_1[\phi_2]$ is satisfied by a pixel x if there is a pixel y, a path π and an index ℓ such that $\pi_0 = x$, $\pi_\ell = y$, y satisfies ϕ_1, and all the intermediate pixels $\pi_1, \dots, \pi_{\ell-1}$ (if any, hence the notation for optional [...]) satisfy ϕ_2.

Note that $\mathcal{N}\phi$ can be derived from ρ, and viceversa. In this paper we use $\mathcal{N}$ explicitly because of its specific implementation in the tool.

From the basic operators, several interesting notions can be derived, such as:

– *interior* corresponding to the imaging primitive of *erosion*;
– *surroundedness*;
– *contact* between regions (see also [10]) denoted as $\mathcal{T}$ (from 'touch').

Concerning region based analysis, the *discrete* Region Connection Calculus RCC8D of [18] has been encoded in a variant of SLCS with operators on sets of points instead of single points [10].

In `ImgQL`, the input language of `VoxLogicA`, these operators have their own syntax. In particular, in the context of the present work we will use:

– `|`, `&`, `!` for the boolean operators disjunction, conjunction and negation, respectively;
– `N` for the near operator;
– `I` for the interior operator, where `I (x) = !(N !x)`;
– `mayReach(x,y)` for the reachability operator $\rho \, x \, [y]$;
– `touch(x,y) = a & mayReach(y,x)` for the contact operator;
– `.<= , .>=` and so on for constraints involving constant bounds (on the side of the dot) and pixel attributes such as intensity or colour.

The tool `VoxLogicA-GPU` is a global, explicit state model checker for SLCS, aiming at high efficiency and maximum portability. `VoxLogicA-GPU` is implemented in FSharp, using the Microsoft dotnet infrastructure, and exploiting the imaging library SimpleITK and the portable GPU computing library OpenCL[3].

Efficiency is one of the major challenges, as outperforming the CPU implementation of `VoxLogicA` [4] inherently means designing in-GPU imaging primitives faster than the state-of-the-art library ITK. The focus of the first release of `VoxLogicA-GPU` has been on demonstrating its scalability. The tool takes as input an `ImgQL` specification and automatically creates a computing pipeline, consisting of a set of tasks to be run. Each task corresponds to a basic primitive (including SLCS operators, and basic imaging primitives such as thresholds). Each arrow denotes a dependency between two tasks, namely that the output of the source task should be fed as the input to the target one. Such task graph is then sent to the GPU, exploiting the so-called *events* mechanism of OpenCL

---

to describe the dependencies. The pipeline is then run on the GPU, respecting the input/output dependencies created by the events mechanism, but otherwise without a specified order.

3   Experimental Setup

The huge speedup attained with the switch to GPUs is theoretically capable of processing video streams in real time. Our work aims at verifying up to which extent this is true using `VoxLogicA-GPU`, and at establishing preliminary use cases for experimentation. Video streams are composed of several frames, each one capturing an instant of a whole scene.

Our envisaged future applications mainly concern two scenarios: that of medical imaging, as mentioned in the introduction, and the field of smart transportation (see e.g. [12]). However, first of all a preliminary study is needed, in order to assess effectiveness of the proposed method, e.g. in terms of complexity of logic formulas, interesting logical operators, and so on.

To this aim, we propose an example based on a video game, namely *Pac-Man*. We provide here a detailed description of the experimental setting, including how execution times are measured, and why we chose to use a footage of a video game as a benchmark.

Pac-Man is a famous 2D video game, released in 1980 by Bandai-Namco. The main character is Pac-Man, represented by a yellow circle, whose goal is to eat all the dots in a maze, avoiding to be captured by the coloured ghosts (Blinky, Pinky, Inky and Clyde) following him. Pac-Man can also eat *energiser pellets* (bigger, blinking dots on the same path as the smaller dots) to gain in turn the ability to catch ghosts. Once caught by Pac-Man, ghosts will re-appear in the centre of the maze. There are various reasons why this simple video game represents an interesting benchmark. The topology of the space remains the same over time, but there are several entities appearing and disappearing. These entities can be effectively specified via a declarative procedure. For this reason, the analysis does not require any kind of preprocessing: video footages have been recorded and split into multiple frames in the `png` format, and then processed directly using `VoxLogicA`.

The `ImgQL` specification of the game elements is shown in Specification 1. Pac-Man is yellow. This is specified by thresholds on the intensity of the red, green and blue components of each pixel. Similarly, for the ghosts and the dots and energiser pellets.

As can be observed in a typical video frame of the game (see Fig. 1), any auxiliary lives of Pac-Man are indicated by small Pac-Man images at the bottom of the frame. Pac-Man itself is inside the maze, so sufficiently far from the border of the frame. In `ImgQL` 'border' is an atomic proposition that holds at all pixels at the extremes of the image. So, Pac-Man itself can be characterised by satisfying the formula `pacmans`, but being sufficiently far from the border, i.e. not touching an area composed of pixels that are at a distance of less than 12 pixels from the border. The `ImgQL` specification is shown in Specification 2.

ImgQL Specification 1: Pac-Man game elements.

```
 1 //find all characters and other elements
 2 let pacmans = (252 .<= red(img)) & (252 .<= green(img)) &
 3                   (10 .>= blue(img))
 4 let blinky = (250 .<= red(img)) & (10 .>= green(img)) &
 5                   (10 .>= blue(img))
 6 let pinky = (250 .<= red(img)) & (175 .<= green(img)) &
 7                   (190 .>= green(img)) & (245 .<= blue(img))
 8 let inky = (15 .>= red(img)) & (240 .<= green(img)) &
 9                   (240 .<= blue(img))
10 let clyde = (240 .<= red(img)) & (190 .>= green(img)) &
11                   (165 .<= green(img)) & (90 .>= blue(img))
12 let dotsAndPellets = (240 .<= red(img)) & (200 .>= green(img)) &
13                   (175 .<= green(img)) & (190 .>= blue(img)) &
14                   (165 .<= blue(img))
```

ImgQL Specification 2: Active Pac-Man.

```
1 let N12 (x) = N N N N N N N N N N N x
2 let pacman = (pacmans & !touch(pacmans, N12 (border)))
```

Pellets and dots have the same colour, but pellets are larger than dots. This information can be used to distinguish them. Taking the area characterised by `dotsAndPellets` and shrinking this area in several steps using the interior operator results in an area comprising only the inner parts of the bigger pellets. The pellets themselves can then be retrieved by extending the thus obtained area by pixels near the remaining area (i.e. centres of the pellets) within 4 steps. This is shown in the second line of Specification 3. The dots are those pixels satisfying `dotsAndPellets` that are not satisfying `pellets`. Figure 1.3 shows the result of `dots`, and Fig. 1.4 shows the result of `pellets`. As we will see, we need to introduce the concept of "uninterrupted paths of dots", specified via the formula `auxDots`, consisting of all pixels that are at most 13 steps (13 times N) from a dot (including dots themselves). Figure 1.5 shows all pixels satyisfing `auxDots`.

ImgQL Specification 3: Dots and pellets.

```
1 let N13 (x) = N N N N N N N N N N N N N x
2 let pellets = N N N N I I I dotsAndPellets
3 let dots = N N (dotsAndPellets & !pellets)
4 let auxDots = N13 (dots)
```

The key points of the game constitute interesting spatial properties. Let's focus on the behaviour of ghosts: each of them will try to catch Pac-Man,

following different routes. We can specify the situation in which Pac-Man is caught by a ghost using `ImgQL` as follows:

```
touch(pacman, blinky) | touch(pacman, pinky)
| touch(pacman, inky) | touch(pacman, clyde)
```

Another significant property, involving reachability, is the fact that Pac-Man can follow a path of dots and reach a pellet (recall that for a game level to be completed, Pac-Man is required to eat all dots in the maze):

```
touch(pacman, mayReach(pellets, auxDots))
```

Intuitively, the property is true if Pac-Man "touches" the beginning of a path of dots leading to a pellet. This is shown in Specification 4. In the second line of the specification the operator `mayReach` is used, denoting the ρ-operator of SLCS. `auxReach` is satisfied by all pixels of pellets and paths formed by enlarged dots as specified in the last line of Specification 3. In the last line one only needs to check whether Pac-Man touches the pixels identified by `auxReach`.

ImgQL Specification 4: Pac-Man reaches pellets and Pac-man caught by ghost.

```
1 let auxReach = mayReach(pellets, auxDots)
2 let result = touch(pacman, auxReach)
3 let caught = touch(pacman, blinky) | touch(pacman, pinky) |
  touch(pacman, inky) | touch(pacman, clyde)
```

This concludes the `ImgQL` specification of the spatial analysis for Pac-Man. The intermediate results of all definitions can be saved separately and numerical information on each frame can be printed as shown in Specification 5. This is how the images in Fig. 1 have been obtained.

ImgQL Specification 5: Saving and printing.

```
1 save "00_pacman.png" pacman
2 save "01_dots.png" dots
3 save "02_pellets.png" pellets
4 save "03_auxDots.png" auxDots
5 save "05_result.png" result

6 print "pathToPellets $NUMFRAME" volume(result) .>. 0
7 print "pacmanCaught $NUMFRAME" volume(caught) .>. 0
```

In order to check the results for each frame, a simple Python script has been written. The script accepts as input a specification fragment such as the ones we have shown so far, and generates a (possibly very large) `VoxLogicA` specification that repeats the given fragment for each video frame, loading each time a different one. In order to identify the result of the analysis on different frames, the placeholder (`$NUMFRAME`) can be used when naming the result values (see also the last line in Specification 5). This placeholder will be replaced with the number of the frame by the Python script while generating the specification. Note also that the result can be both saved as an image or printed as a boolean value. The boolean image *False* corresponds to a completely black image, so we can print the truth value of the formula `volume(resultImage) . > . 0`. As the volume of an image is the sum of its non-zero components, the result will clearly mirror the truth value of the formula. All tests were performed on a workstation equipped with:

– An Intel(R) Core(TM) i9-9900K CPU @ 3.60 GHz (16 cores, 32 threads);
– An NVIDIA GeForce RTX 3080 1785 MHz, 10 GB global memory.

Each test has been run 5 times, in such a way to get an approximate average running time. In our experimental setup, execution times cannot be measured directly by just running the specification. The fact that our video has been expanded to single frames on disk makes input/output times dominate over computation, thus rendering it impossible to measure in a direct way how much time exactly is spent on computation. To overcome this, `VoxLogicA` and `VoxLogicA-GPU` have been modified to accept a command line switch that will let the tool load just one image and reuse it multiple times. Note that this methodology is only used to measure performance and gives a good approximation of the processing time, whereas the loading time is negligible, as only one image is loaded. By carefully choosing, as the representative image, one that exhibits a reasonable complexity, the obtained measurements are representative enough to get a sensible indication of the total computation time. In future work, we will cater for loading video files directly, and unpacking each video frame in memory using optimized (possibly in-hardware) functionality, so that on the one hand, loading of single frames from disk will no longer affect video processing in `VoxLogicA-GPU`, and on the other hand the model checking result will be correct. By the above considerations, we expect the obtained performance to be in line with the measurements that we present in this paper.

4   Results

In this section, we report the results of our experiments. We recall that both the CPU and the GPU versions are tested against *contact* and *reachability* formulas, as described in the previous section. In the considered frame (Fig. 1.1), there is a way for Pac-Man to reach the bottom-right pellet by only eating dots, thus the result image will highlight Pac-Man.

We restricted the tests to a benchmark of 150 frames. This is due to the fact that the CPU version of `VoxLogicA` does not feature any kind of memory management mechanism, thus it suffers from memory limits (on the other hand the memory management mechanism embedded in `VoxLogicA-GPU` allows for larger tests to be run). Videos are RGBA videos, with a resolution of 1600×1200 and 30 fps, namely, 150 frames capture circa 5 s. In Table 1, execution times for the above mentioned formula are reported.

Table 1. Comparison between CPU and GPU execution times, reported in milliseconds
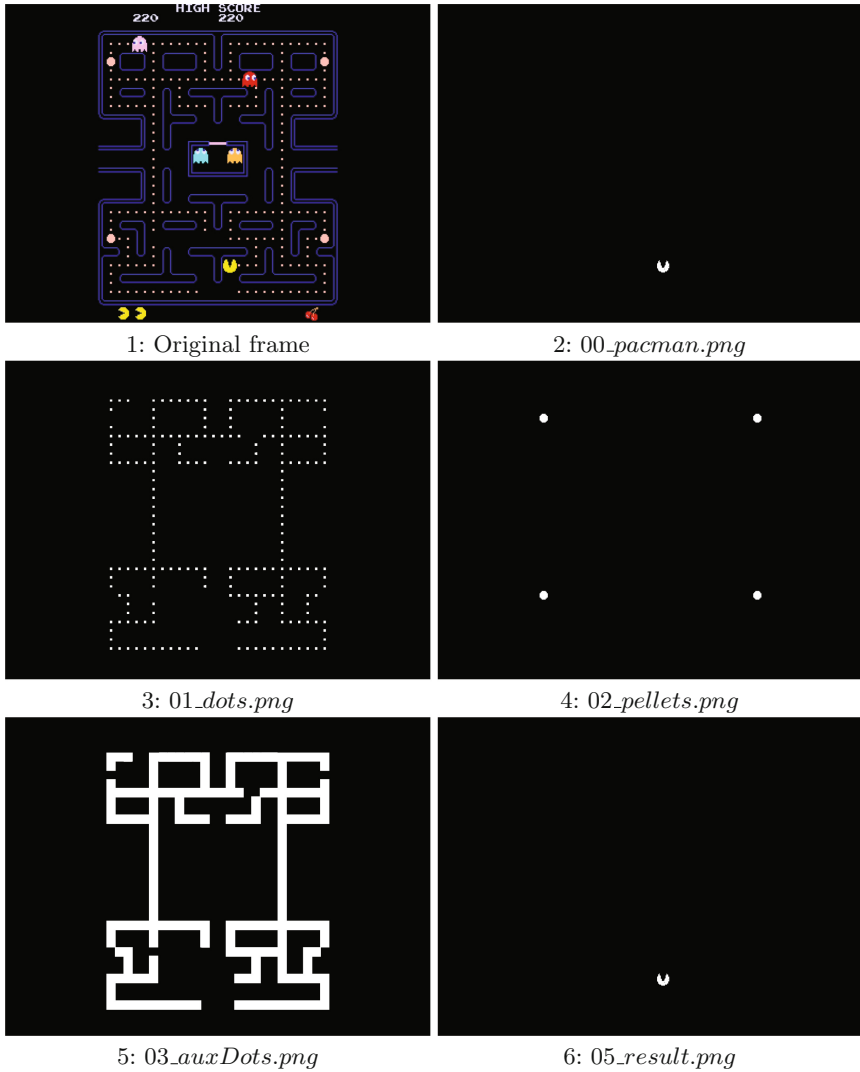
Execution time of sample formulas (150 frames)		
Formula	GPU	CPU
Pac-Man is caught by a ghost (`caught`)	3,800 ms	12,200 ms
Pac-Man can reach a pellet via dots (`result`)	3,000 ms	54,000 ms

The comparison of results shows that using GPUs enable real-time analysis. In this case, we load a pre-recorded video, but our results suggest that monitoring video streaming might be feasible using our technology. It is also interesting to note that the more involved formula actually requires less time on GPU than the simpler collision detection one. This type of results, that may be counter intuitive, depends on the operators involved in the computation:

- in the reachability formula `result`, connected components are computed only once per frame, namely to check if the desired path exists, and "proximity" operators, like Interior and Near, are massively used to find dots and pellets and to enlarge them, in such a way that they form a path. GPUs are more beneficial to this kind of operators, as devices' parallelism can be exploited at its best.
- in the collision detection formula `caught`, connected components must be computed four times for each frame; in fact, we must compute `touch` (which requires computing Connected Components) between Pac-Man and each ghost. Such computation affects the performance of the GPU, as multiple kernels must be called and randomised write and read on the global device memory are required.

Despite these considerations, the execution time stays far below the required real-time threshold. Note that currently it is not possible to state spatio-temporal properties in `ImgQL`, and no temporal analysis is performed in `VoxLogicA`. We leave the possibility to add temporal operators as a future work, possibly still exploiting GPUs.

One may have noted that step 5 in Fig. 1 could cause some false positives. This is due to the fact that enlarging dots using the Near operator causes the paths to touch each other in proximity of pellets. This can be actually avoided

1: Original frame

2: 00_*pacman.png*

3: 01_*dots.png*

4: 02_*pellets.png*

5: 03_*auxDots.png*

6: 05_*result.png*

Fig. 1. Analysis of a video frame (1). Using thresholds, we isolate Pac-Man (2), dots (3) and energy pellets (4). We then enlarge dots in such a way that they touch each other, thus forming a path (5). The result for the formula *Pac-Man can reach a pellet via dots* represents Pac-Man itself (6), as it touches a path leading to a pellet.

by replacing Near with a distance operator based on Euclidean distance (see Fig. 2), whose implementation is currently in progress on VoxLogicA-GPU.

Another interesting point to note is GPU memory and cores occupation. Figure 3 shows that the GPU load is actually quite low. The *nvtop* tool, which is used to monitor processes currently running on GPU, shows that the occupation
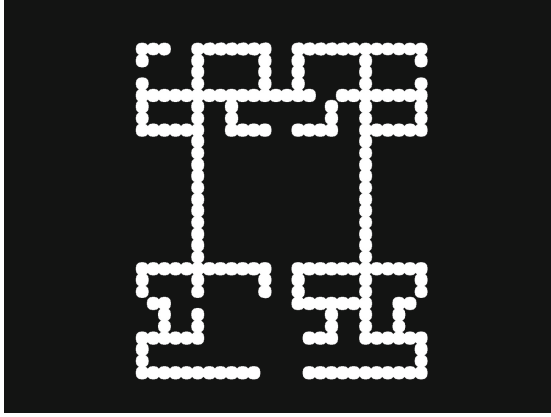
Fig. 2. Path found via the Euclidean distance operator.
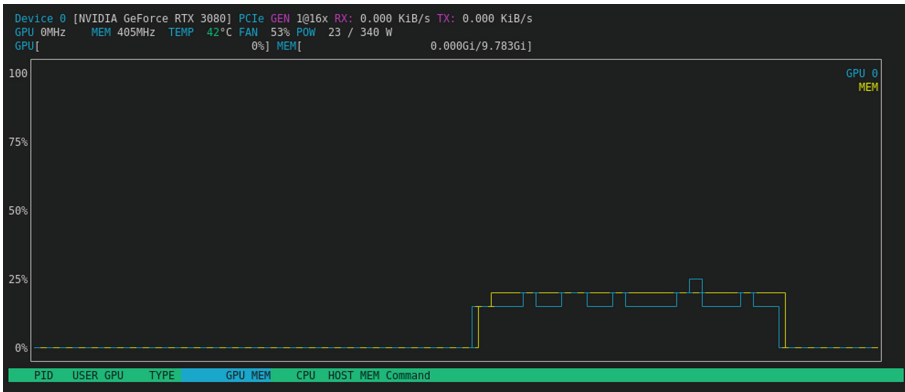
Fig. 3. Memory and cores occupation on GPU, while processing 300 frames.

is circa 25% for both memory and cores, suggesting that the GPU may be used to process larger images and benchmarks, and possibly to monitor 3D simulations.

5   Conclusions

In current work-in-progress, the results obtained in [7] have been vastly improved (ranging from a speed-up of 50× on case studies to 500× on artificial benchmarks). The aim of such optimisation was initially focused on enabling interactive calibration of analysis parameters in ImgQL specifications by end-users of VoxLogicA, and a faster analysis development cycle by domain experts, also in conjunction with a dedicated user interface. The latter is currently being developed taking also cognitive aspects of usability into account (see [6]). However, our preliminary experiments confirmed that this encouraging result can also be

used for a real time analysis of video streams. Despite the fact that temporal operators are currently not available in `VoxLogicA`, it is possible to state and check interesting properties, and this can be done on high resolution videos, in real-time. Furthermore, we showed that the GPU is far from being stressed by the task, thus we expect the speedup to be even larger in case of higher quality videos, 3D animated simulations, or more complex properties.

Still, a large amount of execution time is devoted to I/O operations (in particular loading frames as `png` files). Therefore, a major goal of future work will be to provide `VoxLogicA` with a video loader, possibly exploiting the state-of-the-art OpenCV library[4].

As mentioned above, another goal is to implement temporal operators and a temporal model checking algorithm in `VoxLogicA-GPU`. This will widen the set of properties amenable to spatio-temporal model checking of video-streams, opening possibilities of new scenarios.

Our experiments will also be useful in future work for expanding the logical language with dynamic binding predicates, giving the language the ability to capture the concept of a "new entity", different from all the previously known ones which appears in a video stream at a given instant, and that may reappear later. In our physical simulation example, binding can be used e.g. to answer queries like "was Pac-Man caught twice by the same ghost x", independently from the moment in which x appears for the first time, and the identity of x.

References

1. Banci Buonamici, F., Belmonte, G., Ciancia, V., Latella, D., Massink, M.: Spatial logics and model checking for medical imaging. Int. J. Softw. Tools Technol. Transfer **22**(2), 195–217 (2019). https://doi.org/10.1007/s10009-019-00511-9
2. Bartocci, E., Bortolussi, L., Loreti, M., Nenzi, L.: Monitoring mobile and spatially distributed cyber-physical systems. In: Talpin, J., Derler, P., Schneider, K. (eds.) MEMOCODE 2017, pp. 146–155. ACM (2017)
3. Belmonte, G., Broccia, G., Ciancia, V., Latella, D., Massink, M.: Feasibility of spatial model checking for nevus segmentation. In: 2021 IEEE/ACM 9th International Conference on Formal Methods in Software Engineering (FormaliSE), pp. 1–12 (2021). https://doi.org/10.1109/FormaliSE52586.2021.00007
4. Belmonte, G., Ciancia, V., Latella, D., Massink, M.: `VoxLogicA`: a spatial model checker for declarative image analysis. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11427, pp. 281–298. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17462-0_16
5. Berkovich, S., Bonakdarpour, B., Fischmeister, S.: GPU-based runtime verification. In: IPDPS 2013, pp. 1025–1036. IEEE Computer Society (2013)
6. Broccia, G., Milazzo, P., Ölveczky, P.C.: Formal modeling and analysis of safety-critical human multitasking. Innov. Syst. Softw. Eng. **15**(2), 169–190 (2019). https://doi.org/10.1007/s11334-019-00333-7
7. Bussi, L., Ciancia, V., Gadducci, F.: Towards a spatial model checker on GPU. In: Peters, K., Willemse, T.A.C. (eds.) FORTE 2021. LNCS, vol. 12719, pp. 188–196. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-78089-0_12

---

[4] See: https://opencv.org/.

8. Ciancia, V., Latella, D., Loreti, M., Massink, M.: Specifying and verifying properties of space. In: Diaz, J., Lanese, I., Sangiorgi, D. (eds.) TCS 2014. LNCS, vol. 8705, pp. 222–235. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44602-7_18

9. Ciancia, V., Latella, D., Loreti, M., Massink, M.: Model checking spatial logics for closure spaces. Log. Methods Comput. Sci. **12**(4) (2016)

10. Ciancia, V., Latella, D., Massink, M.: Embedding RCC8D in the collective spatial logic CSLCS. In: Boreale, M., Corradini, F., Loreti, M., Pugliese, R. (eds.) Models, Languages, and Tools for Concurrent and Distributed Programming. LNCS, vol. 11665, pp. 260–277. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21485-2_15

11. Ciancia, V., Belmonte, G., Latella, D., Massink, M.: A hands-on introduction to spatial model checking using VoxLogicA. In: Laarman, A., Sokolova, A. (eds.) SPIN 2021. LNCS, vol. 12864, pp. 22–41. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84629-9_2

12. Ciancia, V., Gilmore, S., Grilletti, G., Latella, D., Loreti, M., Massink, M.: Spatio-temporal model checking of vehicular movement in public transport systems. Int. J. Softw. Tools Technol. Transfer **20**(3), 289–311 (2018). https://doi.org/10.1007/s10009-018-0483-8

13. Ciancia, V., Latella, D., Massink, M., Paškauskas, R., Vandin, A.: A tool-chain for statistical spatio-temporal model checking of bike sharing systems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9952, pp. 657–673. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47166-2_46

14. Ciancia, V., Latella, D., Massink, M., de Vink, E.P.: Towards spatial bisimilarity for closure models: logical and coalgebraic characterisations. CoRR abs/2005.05578 (2020). https://arxiv.org/abs/2005.05578

15. Ma, M., Bartocci, E., Lifland, E., Stankovic, J.A., Feng, L.: A novel spatial-temporal specification-based monitoring system for smart cities. IEEE Internet Things J. **8**(15), 11793–11806 (2021). https://doi.org/10.1109/JIOT.2021.3069943

16. Neele, T., Wijs, A., Bošnački, D., van de Pol, J.: Partial-order reduction for GPU model checking. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 357–374. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46520-3_23

17. Osama, M., Wijs, A.: Parallel SAT simplification on GPU architectures. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11427, pp. 21–40. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17462-0_2

18. Randell, D.A., Landini, G., Galton, A.: Discrete mereotopology for spatial reasoning in automated histological image analysis. IEEE Trans. Pattern Anal. Mach. Intell. **35**(3), 568–581 (2013). https://doi.org/10.1109/TPAMI.2012.128

19. Wijs, A., Bošnački, D.: Many-core on-the-fly model checking of safety properties using GPUs. Int. J. Softw. Tools Technol. Transfer **18**(2), 169–185 (2015). https://doi.org/10.1007/s10009-015-0379-9

20. Wijs, A., Neele, T., Bošnački, D.: GPUexplore 2.0: unleashing GPU explicit-state model checking. In: Fitzgerald, J., Heitmeyer, C., Gnesi, S., Philippou, A. (eds.) FM 2016. LNCS, vol. 9995, pp. 694–701. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48989-6_42