Synthetic tracking for orbital object detection in LEO

Josua Zscheile[(1)], Paul Wagner[(1)], Raoul-Amadeus Lorbeer[(1)], Benjamin Guthier[(2)]

[(1)] German Aerospace Center (DLR), Institute of Technical Physics, Pfaffenwaldring 38-40, 70569 Stuttgart, Email: paul.wagner@dlr.de
[(2)] Technische Universität Dresden, Faculty of Computer Science, 01062 Dresden, Germany

ABSTRACT

Passive optical systems are limited in their capabilities to detect and track unknown objects in Low Earth Orbit (LEO), especially if these are very faint. The established streaking methods that utilize long exposure times suffer from loss of signal due to the fact that the photons reflected by the object are distributed over several dozens to hundreds of image pixels and form a streak. The last years have seen a rebirth of "Synthetic Tracking" (ST). This approach mitigates the effect of the photon distribution by taking many pictures with short exposure times and adding up the signal of these images. This leads to the light being focused on a few pixels and less noise from surrounding pixels. We propose the use of ST for the detection of LEO objects on graphics hardware and evaluate its real-time capabilities for future use in conjunction with a ranging laser to obtain accurate trajectory information.

1. MOTIVATION

At the time of this work radar astrometry is the primary ground-based method used for detection of LEO objects. Radar telescopes have reportedly been able to detect objects with diameter as small as 2 mm [1]. Currently around 17,000 objects are being observed consistently [2], the smallest of which have a diameter of 10 cm.

In comparison, (passive) optical methods are not that prominent for ground-based detection of LEO objects. They are usually used for objects in GEO and farther away because active systems like radar do not have the necessary range for objects in GEO and beyond. In LEO optical systems have the potential to complement radar in the task of debris detection due to the lower costs compared to potent radar arrays and some objects being more reflective in the visible spectrum than in the RF spectrum.

While the optical detection of very bright objects like the ISS is trivial, the task is more complex for very faint objects that are far from being detectable by the human eye. The dominant method of the past decades and for the time being is called streak detection or "streaking".

1.1 Streaking

For streak detection of LEO objects a camera system with large field of view (some $(°)^2$) is used with long exposure times (around 1s, depending on the field of view and the orbit to observe). LEO objects in the resulting images form bright lines, also called streaks, because their apparent movement over the exposure time spans multiple pixels.

The exposure time has to be carefully chosen. Longer exposure times lead to longer streaks which are more easily detectable by edge detection algorithms and contain more robust information about the orbit parameters compared to short streaks. On the other hand, longer streaks have a higher probability of having "start" or "end" point (or both) beyond the field of view, in which case a trajectory estimation can only have a lower limit in terms of object speed. Additionally, from one exposure the directionality of the movement cannot be determined.

The signal on streaking images is distributed over many pixels, degrading the signal strength in comparison to cameras in a tracking mount, following a known object, where the light reflected by the object is collected in a few pixels.

1.2 Synthetic Tracking

Synthetic Tracking (ST) simulates the behaviour of cameras operated in tracking mode with series of short exposure images. In contrast to these cameras, it can detect new objects and track multiple objects at the same time. The algorithm will be discussed in detail Chapter 3.

An important value when comparing different techniques is the signal-to-noise ratio (SNR). SNR in general is a measure of signal intensity against the noise intensity. For astronomic images where the observed objects can be considered to be a non-resolved point source the SNR is usually measured as the peak value of the light source divided by the background mean value. Since the mean value of noise is offset by background intensity, this is subtracted beforehand. This leads to the definition of SNR as follows [3]:

$$SNR(S) = \frac{S}{\sqrt{B}} \quad (1)$$

where S is the peak signal intensity and B is the background mean intensity.

For streaks on the image the photons of the signal peak are additionally distributed over multiple pixels along the streak [4]. The signal thus becomes

$$S' = \frac{S}{l} \quad (2)$$

with l being the streak length in pixel. As the background remains the same in both scenarios, the SNR of a static object can be directly compared to the SNR of the moving object by the formula

$$\frac{SNR(S)}{SNR(S')} = l \quad (3)$$

due to more pixels with more noise in the streaked case. That means, depending on streak length, that ST shows promise to find objects orders of magnitude dimmer than streaking methods can detect. In addition, directional ambiguity is not present and the start and end points of objects can be determined much more accurately, leading to better trajectory estimation when compared to streaking.

2. RELATED WORK

The first mention of Synthetic Tracking [4] explores the feasibility of this algorithm for detection of Near-Earth Asteroids (NEAs). To achieve this, the authors stack 60 images recorded by a 200 inch telescope with 500 ms exposure time. They achieved SNR higher by 20 to 40 compared to streaking methods.

In 2017 some of the same authors [5] proposed the use of ST on multiple CubeSats to radically increase the number of known and continuously tracked NEAs. In this configuration they plan to stack 80 images of 10 s exposure time recorded by an on board 10 cm telescope.

In 2018 the JPL scientists reported again on their successes with ST on a ground-based 40 inch telescope [6] as well as on board the SkySat-3 satellite [7].

While the authors promoted the potential in terms of real-time processing, at the time of this work their presented detections all ensued in post-processing. Furthermore, NEAs can be observed with large telescopes because of their low apparent velocity.

The so-called Shift-and-Add method (to be described below) that is the basis for the ST strategy is much older. For speckle astronomy, the degrading atmospheric influences are resolved by taking many (not necessarily digital) exposures of a celestial object, shifting them with respect to each other so that the brightest spots aligns, and adding their intensity values up [8]. If only the most promising exposures are added up, the approach is often called "Lucky Imaging" [9]. First validation of far away, unresolved binary or multi-star systems from speckles was a typical use case of these approaches. SAA is also used for validation of an orbiting object [4], where the orbit of an object is known at least broadly and the observation is meant to refine and confirm prior observations. In this case the amount of shifts that have to be calculated is much lower due to prior knowledge than in the case presented here, where goal is to achieve new detections and the whole spectrum of angles and velocities has to be searched in.

In contrast to all these contributions, we propose the use of Synthetic Tracking for detection of space debris in LEO and show, that the algorithm has the capabilities of being optimized to a degree where real-time detection and tracking becomes possible. The former has been shown to work very recently [10]. This offers the possibility to use other instruments, e.g. ranging lasers, to refine orbital measurements on the same overpass to a point where the object can be continuously tracked in future overpasses, not getting lost again due to too high uncertainty in the orbital parameters.

3. METHOD

3.1. The Synthetic Tracking Algorithm

Synthetic Tracking is based on the Shift-and-Add (SAA) algorithm. It takes as input a stack of n+1 images $S = \{I_0, I_1, \ldots, I_n\}$ (usually video frames) and a shift vector $v = (v_x, v_y)$ and returns a shifted image I_{SAA}.

The input images are ordered consecutively with respect to time, ideally with a set framerate. The vector v denotes a proposal for the apparent movement vector of one or more unresolved (point-like) objects through the image stack. The value for a pixel (x, y) in the shift image I_{SAA} for a shift vector $v = (v_x, v_y)$ is calculated as

$$I_{SAA}(x, y, v_x, v_y) = \sum_{i=0}^{n} I_i(x + \frac{iv_x}{n}, y + \frac{iv_y}{n}) \quad (4)$$

An object moving "through" the image stack with directional velocity v results in a high pixel value in I_{SAA}, as all the faint signals on the single images, possibly not distinguishable from noise, add up. Doing this for every pixel results in the complete shift image.

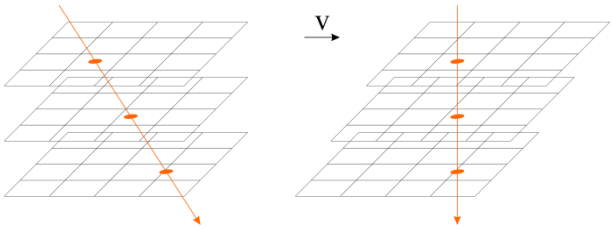

Figure 1. Illustration of Synthetic Tracking principle

Synthetic Tracking calculates the shift images for every sensible shift vector v. The choice of the vectors is based on how fast objects in orbit are expected to move with respect to the camera setup. Our reference system is an Andor Zyla sCMOS 5.5 with a Zhongyi Mitakon Speedmaster 85 mm f/1.2 objective pointed at zenith, a minimum orbit height of 400 km and a maximum orbit height of 1200 km. This system has a field of view of 9.4° by 11.1°. Assuming a circular orbit, this means that the maximum length of the shift vector $|v|_{max}$ = 251 px, and the minimum $|v|_{min}$ = 79 px at the highest resolution of 2560x2160 px. At this image resolution, each pixel spans 15.8" by 15.8"

3.2 Realisation on GPU

Adapting an algorithm for efficient use of graphics cards resources is a task that requires special care in order to weigh the different kinds of memory, processing units and their capabilities, capacities, throughput and bandwidths. Our implementation uses the CUDA computing platform for NVIDIA graphics cards in C++. High efficiency of calculations on graphics cards is achieved with a high degree of parallelization. The task should be subdivided into many small sub-tasks, optimally in such a manner that the sub-tasks perform the same calculations on different input to as high an extent as possible. This is called the Single Instruction Multiple Data (SIMD) paradigm. When one such sub-task is mapped on one computation thread on GPU, this consequently follows the Single Instruction Multiple Threads (SIMT) architecture of graphics cards.

The instructions for every thread are defined in a kernel that is written like a function in C. Threads are bundled in blocks. A set of up to 32 threads in a block can be executed simultaneously, because each Processing Block has 32 cores; this set is called a warp. In our implementation each thread is assigned a pixel in the first image of the stack. The set of all blocks span the whole image.

Threads in a warp can only perform the same operation in at the same time, if they all execute the same instruction, following the SIMT paradigm. If threads diverge, they have to be executed consecutively instead. This is mostly the case, if the controlling condition of some *if* or *switch* instruction or some kind of loop includes a component of the thread ID. This loss of thread concurrency needs to be avoided.

The programmed kernel needs to be revised instruction for instruction carefully. Considerable boosts in performance can be achieved by avoiding costly operations, type conversions and repeating unnecessary operations in loops. An example of expensive operations that can be replaced is divisions; they can be replaced by multiplications with the reciprocal, if the divisor is the same in multiple instructions.

A graphics card usually has multiple Streaming Multiprocessors (SMs) who in turn have multiple Processing Blocks (PBs). While only one warp can be active at a time on a PB, loading and storing times are effectively masked, because a SM can have multiple warps in execution per PB, the inactive ones of which are waiting for load or store operations. The memory of a graphics card can be divided into three physically different entities: global memory which all SMs can access, Shared Memory private to one SM and registers private to single threads. The access times and sizes scale accordingly; global memory is slow to access but large, shared memory is rather fast but limited, and registers are accessed very fast but can only store very limited data.

Our implementation exploits these features. The stack of input images, while too large to reside in Shared Memory, is declared as a layered stack of 2D textures which reside in global memory. These textures are read-only arrays that, when addressed, get cached in the much faster local cache instead of the slower, larger global cache. Access to a value in a texture is handled by specialized texture units, who at the same time can linearly interpolate the requested value, if an intermediate-pixel position is needed. Lastly, texture caching is performed in both directions of the texture instead of only one direction for general global memory accesses. This feature alone results in a speed increase by a factor of four over an equal global memory only implementation, as shown in chapter 4.2.

Each thread in our implementation calculates all shift values $I_{SAA}(x, y, v_x, v_y)$ for fixed x, y and v_y, iterating through all input images and a subset of all v_x. We iterate through all of the v_x for each image. Because each image forms its own texture layer, the input data requested by the threads is cached with a few accesses, and all calculations that need this data are performed, before the next texture has to be loaded. The specialized texture units perform the linear interpolation very efficiently which is usually necessary before adding a value.

The intermediate results of this sum have to be stored. It is not efficient to store them in global memory, and registers have too little capacity in the general case. As an aside, they cannot be used for dynamic allocation,

which is useful in many cases, including the ST algorithm. Instead, we store these in faster shared memory. The amount of shared memory needed depends on the amount of v_x processed in one kernel call and the block size, because shared memory needs to be allocated for all threads of a block. Data in shared memory is only stored as long as the block is processed, so the results have to be written to global memory before the kernel execution ends. The choice of block size and amount of v_x calculated in one kernel call have to be carefully balanced to ensure peak performance, because the limited shared memory is allocated per block and the amount of blocks that can be executed on a SM concurrently depends on the fraction of shared memory they occupy.

The main reason why shared memory is faster than global memory is that it is divided into 32 equally sized memory banks, which can access their data independently of each other. The data is split between banks in such a way that consecutive 32-bit words of data are stored on neighbouring banks. Therefore up to 128 Byte of data stored in succession can be accessed at the same time. On the other hand, accessing every m^{th} value where m is a multiple of two results in bank conflicts, meaning that multiple words are stored on one bank, and the accesses have to be serialized, which delays all further execution accordingly.

We have taken special care to avoid bank conflicts by manipulating the order in which data is stored in shared memory. Which addresses are accessed at a given computation step is dependent on the amount of v_x computed and the dimensions of the block. In the general case it is fastest to store the intermediate results of one thread consecutively in memory, followed by the results of the next thread. This way any kernel execution that includes $v_x = 0$ will never lead to bank conflicts. However, with our setup, in cases where $|v_x|$ is divisible by four, another kernel performs better. This kernel stores the results of same v_x consecutively, followed by the results of the next. While this concrete effect may not occur on every system, it indicates that system specific details may have relevant performance impact due optimizations performed during execution on a specific type of GPU.

4. RESULTS

4.1. Detection of LEO objects

For verification we tested our implementation on simulated data that takes into account atmospheric influences by modelling a point-like light source as a two-dimensional Gaussian intensity distribution and Photon shot noise modelled as random Poisson distribution of the intensities as well as background brightness. In accordance to the maximum of 100 fps the model camera can record we chose stack height of 100 images for our analysis. Furthermore, we compute all shift vectors v in increments of 1 ("step size") in their components. A more fine-grained search does not yield in improved results because of the use of linear interpolation, and higher step sizes lead to a rapid loss of accuracy. We leave detailed analysis of the performance of ST with respect to minimum object brightness and according data evaluation strategies for future work. For the results presented we applied a simple algorithm that extracts the maximum intensity of a shift image and returns it along with the pixel position, in the image for every computed shift. In case of an object, this pixel position is the position of the object in the first image of the stack. We then compare all the extracted maxima with each other. For illustration purposes, we additionally subtracted the minimum value of the shift domain from all values. Utilizing this method, objects register as global maxima in the shift vector space for SNR as low as 0.4 reliably, and we have encountered global maxima for objects with SNR as low as 0.1.

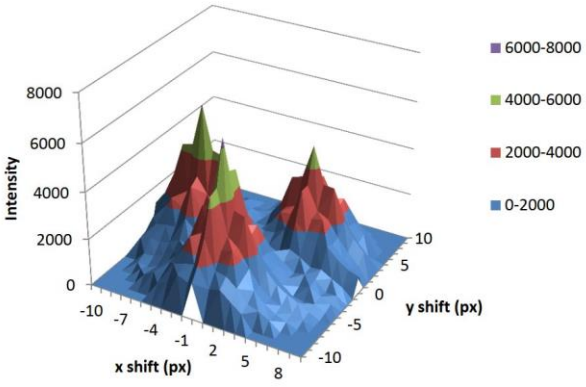

Figure 2. Example diagram of three objects moving with vectors (-7px, 0px) (left), (0px, -7px) (centre) and (5px, 3px) (right), respectively. Noise mean is 132, SNRs are 0.41, 0.41 and 0.37, respectively.

We note that an object usually still registers with the correct position and movement vector, even if noise replaces it as global maximum in shift space. This indicates that correlation with results from input stacks analysed prior or following could result in much lower SNR for detection.

Furthermore we briefly present the behaviour of the maximum based method we chose.

Objects with same brightness show a higher intensity in the shift space the closer they are to the shift component axes, as Fig. 2 shows. The on-axis objects register with both higher intensity in shift space as well as higher SNR in the input data, because their spatial distribution of the intensity always concentrates the light in a pixel's centre for the movement direction. For high brightness regimes (SNR > 0.5) objects moving in a 45° angle to

the axes result in 20% to 25% lower intensity in the shift images when compared to objects moving along the axes. As a result of fewer random (noise) values contributing to the values in the according shift image, higher variance in the values and the selection of the maximum pixel in a large image, noise intensity on the axes in the shift domain is also more pronounced (Fig. 3).

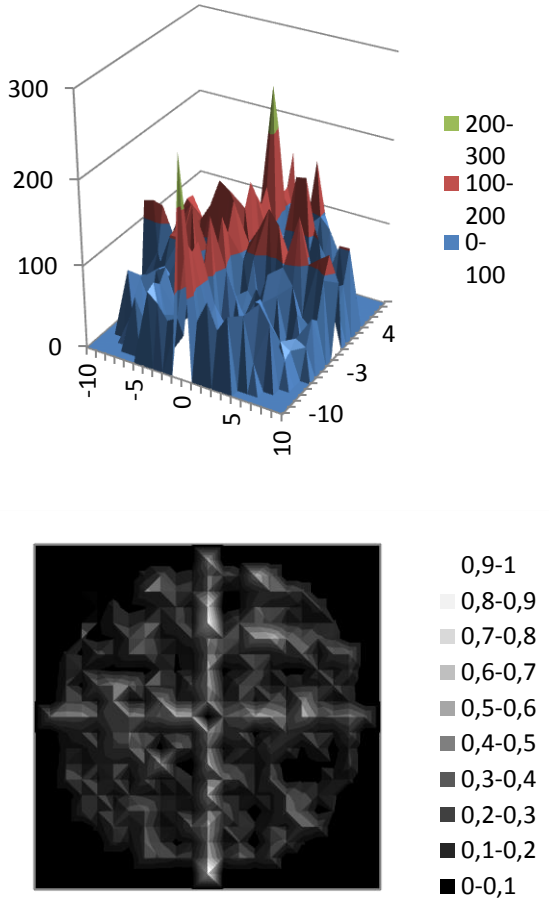

Figure 3. Shift diagram of noise without objects (top). The subtracted minimum value is 13,569.8, background noise mean value is 66. Top view of the normalized shift diagram to illustrate the location of high values (bottom).

Another interesting aspect is the effects of very low influence of atmospheric disturbances (low standard deviation) in combination with a starting and end position of the object on or very near to edges or corners between pixels. Due to the lower signal intensity in these cases and the effects of linear interpolation of the values an object can register up to four peaks in shift domain in neighbouring pixels, with a local minimum, where the ground truth shift should register. The left object in Fig. 4 illustrates this effect; the real velocity vector is located in the middle between both peaks. This effect should however not occur under realistic seeing conditions, and is easy to catch, should it occur.

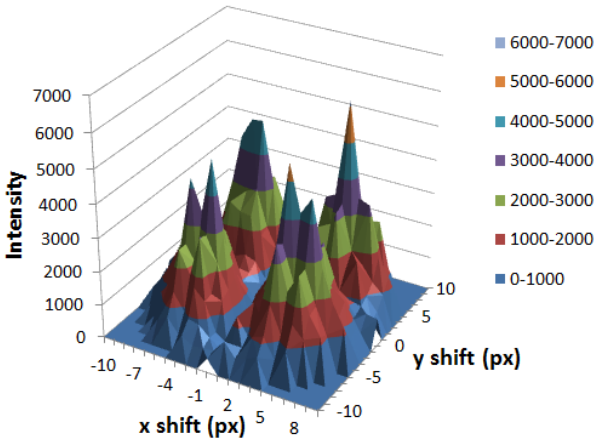

Figure 4. Aliasing effect of four different in-between pixel starting positions ("edge" case). Shown are objects starting at (x, y) (right), (x+0.35, y) (back), (x+0.45, y) (front) and (x+0.5, y) (left) with Gaussian intensity standard deviation of 0.1, background noise of 132 and SNR = 0.85.

We expect future research to find more sophisticated methods to extract objects and their significant parameters from the results ST provides.

4.2 Run time and Real-time capability

The run time evaluation presented was performed on our development system with a NVIDIA Quadro K620 graphics card with 2 GB DDR3 RAM, introduced in 2014 and an Intel Xeon E5-2620 core i7 2.10 GHz CPU with 8GB RAM. Additionally, we performed some analysis on a system with a NVIDIA Quadro P4000 with 8 GB DDR5 RAM from 2017 and an Intel Xeon E3-1270 core i7 3.60 GHz with 32 GB RAM. To account for fluctuations on load and frequency of the graphics card, the results are averages over ten measurements. The effects of different parameter choices are measured against the default configuration presented in Tab. 1.

Frame dimensions	500 x 400 px
Pixel depth	16 bit
Stack Height	100 frames
Number of passes	10
Minimum velocity	1 (px/stack)
Maximum velocity	10 (px/stack)
Step size	1
Resulting total number of shifts per pass	316

Table 1: Parameters for the computation time tests

The optimum run time for the computation of the SAA kernels on the K620 is 403ms; with the kernel that determines the maximum of each shifted image included the time is 438ms. In comparison, the according P4000 runtimes are 62ms and 69ms, respectively. The former system is bound by shared memory bandwidth, while the latter is bound by texture memory bandwidth. Taking the maximum amount of texture fetches of the P4000 as theoretical limit, the computation could be sped up further by a maximum factor of 1.45 (94.9 GTexels/s in our implementation vs. theoretical 137.4 GTexels/s maximum).

Although it is hard to determine run times on systems never tried, we expect at least an equal speed-up over the P4000 with state-of-the-art Titan and RTX graphics cards, as they have 72 SMs (Titan RTX) in comparison to the 14 (P4000) and 3 (K620), DDR6 memory (DDR5 and DDR3, respectively) and higher clock rates (1350MHz over 1227MHz and 1058MHz), among other improvements.

	global memory only	input as texture	Additionally intermediate results in shared memory
Kernel run time (ms)	5,470	1,439	438

Table 2: Run time comparisons exploiting the hardware components of graphics cards to different degrees

Computation time is linearly dependent on image size in pixels, number of images processed (stack height) and number of shift vectors computed (Fig. 5).

Block size y \ Block size x	1	2	4	8	16	32	64
1						467	444
2					484	450	462
4			766	476	466	472	482
8			460	454	489	488	
16		447	441	476	504		
32	462	438	474	499			
64	475	474	503				

Table 3: relation of computation time (in ms) and block size. Computation time includes maxima calculation.

As Tab. 3 shows, the choice of block size is also impactful on performance. In contrast to the other parameters however, there is no obvious mathematical connection between the two.

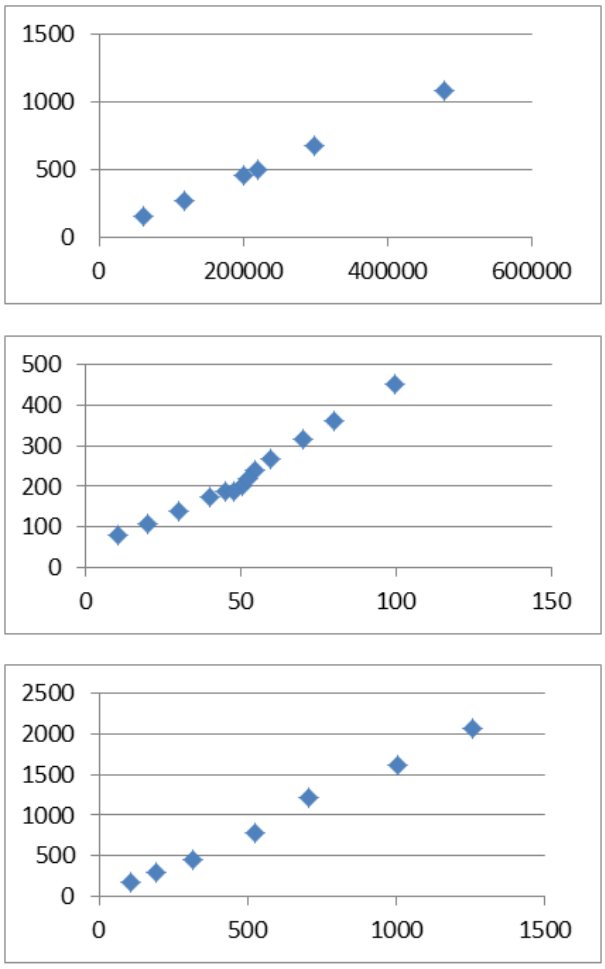

Figure 5. Comparison of image size in px (top) stack height in images (centre) and number of calculated shifts (bottom) to computation time in ms.

It is obvious that total block size (both components multiplied) should be at least 32. That is because the maximum number of threads in a warp is 32, and a warp can never consist of threads of multiple blocks. Therefore no other block sizes apart from 4x4 that are not multiples of 32 were tested. The fastest computation times are situated on the diagonal for a total block size of 64. An explanation for this phenomenon could be that this amount of threads hits the optimum between yields more cache hit than configurations with block size 32, but has less memory usage than bigger block sizes. Finally, apart from some extreme cases, choosing one size smaller than the other has shown to improve performance, with small x component and big y component being slightly better. It cannot be ruled out that these findings are specific to the hardware used, therefore testing for an optimal configuration with other hardware may have an impact of a few percent on computation time.

5. OUTLOOK

The setup leading to the results in Tab. 2 computes 316 shifts of images of size 500x400 px in 438ms. The camera setup that is the real world model for this work captures images of 2560 x 2160 px. That is 27.65 times the amount of pixels. Moreover, all directions and speeds for objects between 400 and 1200 km orbit height have to be calculated. This means to maximum velocity in a stack of frames spanning one second of recording is 251 px/s, the minimum being 79 px/s. The number of shifts that need to be calculated is approximated by the difference of the areas of circles with these radii:

$$A_{circle} = \pi \cdot r^2 \qquad (5)$$

This is only an approximation according to the Gauss circle problem. The exact number can be determined by checking for each point on the grid if its distance to the centre point of the circle is less or equal to the radius. For the values above it is 178272. Compared to the 316 shifts in the benchmarking case this means 564.15 times the amount of shifts need to be calculated. Not taking into account that the graphics card of the development system could not store the input frame stack in global memory, this increase in calculations by a factor of roughly 15,600 would result in a computation time of 114 min for one second of input data. This means that a system aiming to calculate this in one second would need 6832 times the computing power of the graphics card present in the development system. The system with the single P4000 graphics card could do the same in just under 18 minutes. Assuming the same speed up of 6.3 times with a state-of-the-art graphics card, the computation time can be estimated to be less than 3 minutes; 170 of these graphics cards clustered could calculate all shifts in the sensible range. Extrapolating the capabilities of graphics cards into the future, the cards introduced in 2021 should be able to do this in a cluster of 27 cards.

The supercomputer Summit, built in 2014, could run the data processing of an estimated 166 cameras at the same time only using its graphics cards. If we assume good seeing conditions for $1/12^{th}$ of cameras at the same time, it even could support a global network of 2000 cameras (data transmission times notwithstanding).

Most of the scientific community does not currently have these kinds of resources at their disposal. However, data reduction techniques can be used to further reduce run times.

Alternatively to using clusters of graphics cards other measures can be taken to reduce the amount of computations to perform. We implemented a function that calculates shifts only for stripes on the borders of the image, and only in the "inward" directions. The reason for this approach is that in an ongoing observation, objects appear at the edge of the observed area of space and do not need to be found multiple times in the same overpass. The stripe width should be adapted to the maximum velocity an observed object can have; in the model system the fastest possible object would be visible for at least 8 seconds. As such, only a fraction of each image needs to contribute for computation, and only roughly half of the shifts need to be calculated. Setting the stripe width to 502 px (double the maximum apparent velocity) would reduce the computation time by a factor of 0.38 to 43.3 minutes on the development system's graphics card.

Another approach that is often used to reduce the amount of data in astronomy is pixel binning. Binning with a factor n is a technique where the values of n by n pixels are collected into one instead. Binning by two would reduce the frame pixel count to 1/4th and the amount of shifts to compute also to 1/4th (44300 shifts instead of 18272), for a total sixteen-fold saving in computations and computation time. The Quadro K620 used in development could compute this in just over 7 minutes, requiring a factor of 427 for real-time capability, which can be achieved with a few state-of-the-art consumer-grade graphics cards as exemplified above. Of course, the sensitivity (SNR) and accuracy (in pixel positon and speed vector) drop by a factor of two each.

If the computational resources are limited, the shift vector space can be limited to meet real-time requirements. Restricting the search space to a range of orbital directions can be performed e.g. by adding an offset to a much smaller search radius. For example, a circle that spans the whole search band with its diameter has only 23217 shifts that need to be calculated, a speed increase by a factor of 7.7.

Alternatively, the search could be restricted to certain orbit heights, which can be performed by choosing a narrower band of orbital velocities. Fig. 6 (bottom) illustrates this method for orbits of height between 400 and 500km, which translates to velocity vectors of length 200 to 251, decreasing the search space and time by a factor of over 2.7. Of course the possibilities are not limited to a choice of shifts that is easy to select from how the algorithm currently works or how it can be easily adapted. Observing different velocity and direction portions of shift space at the same time is possible with the according hardware.

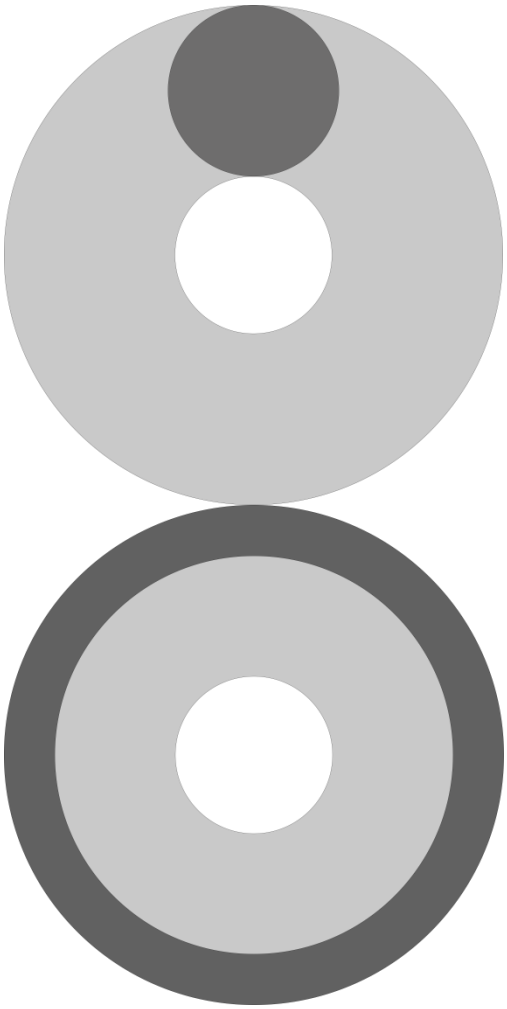

Figure 6. Proportional illustrations of strategies to limit the search to certain parts of shift space using an offset (top) or a narrower band (bottom).

6. SUMMARY

We have shown that Synthetic Tracking is a technology useable for real-time detection and tracking of LEO objects. The necessary hardware to perform continuous real-time brute force search for all possible LEO objects is not commonly available. Consequently, we have shown that data reduction techniques can be used to decrease computation time to real-time capability levels. Furthermore the projected increase in computing power of near future graphics cards over current state-of-the-art models is expected to make such compromises unnecessary in the coming years.

The prerequisite for such high performance is a highly optimized implementation of the algorithm. We have provided guide lines to achieve this level of optimization.

The dominant method of optical detection of faint LEO objects requires expensive optical systems with large aperture telescopes. While not in the focus of this paper, Synthetic Tracking shows promise to increase the performance of single aperture systems, increasing sensitivity and decreasing costs of the optical system.

Synthetic Tracking is a rather novel approach to detection of celestial objects, especially in the fast regime of LEO. There is still much research work to be done. This includes efficient real-time pre-processing of the input data, more sophisticated strategies to evaluate the output and suitable data reduction techniques. When these problems get addressed, Synthetic Tracking has the potential to become the foundation of a high performance, highly scalable global optical debris detection system.

7. REFERENCES

1. Mehrholz, D. et al. (2002). Detecting, Tracking and Imaging Space Debris, In *ESA bulletin 109*, European Space Agency
2. space-track.org Frequently Asked Questions: What are analyst objects?, Online at https://www.space-track.org/documentation#faq (as of 05.12.2018)
3. Lim, S. (2008). Characterization of Noise in Digital Photographs for Image Processing, *HP Laboratories*
4. Shao, M., Nemati, B., Zhai, C., Turyshev, S.G. & Sandhu, J. (2014). Finding Very Small Near-Earth Asteroids using Synthetic Tracking, In *The Astrophysical Journal*, Volume 782, Number 1
5. Shao, M., Turyshev, S.G., Spangelo, S., Werne, T. & Zhai, C. (2017). A constellation of SmallSats with synthetic tracking cameras to search for 90% of potentially hazardous near-Earth objects, In *Astronomy & Astrophysics*, Volume 603, Article A126
6. Zhai, C. et al. (2018). Accurate Ground-based Near-Earth-Asteroid Astrometry using Synthetic Tracking, *arXiv.org e-Print archive*
7. Zhai, C. et al. (2018). Technical Note: Asteroid Detection Demonstration from SkySat-3* B612 Data using Synthetic Tracking, *JPL Publication* 18-1
8. Baba, N., Isobe, S., Norimoto, Y. & Noguchi, M. (1984). Stellar speckle image reconstruction by the shift-and-add method, In *Applied Optics* 24(10):1403-5
9. Tokovinin, A. et al. (01.12.2010). High-resolution imaging at the SOAR telescope, In *Publications of the Astronomical Society of the Pacific*, Volume 122, Issue 898, pp. 1483
10. Shao, M., Trahan, R., Zhai, C., Saini, N. & Turyshev S.G. (2018). Synthetic Tracking on a Small Telescope, Advanced Maui Optical and Space Surveillance Technologies Conference