Optimizations of a Generic Holographic Projection Model for GPU's

Mark Voschezang[1,2][0000−0001−5585−7907] and Martin Fransen[2][0000−0002−0932−2984]

[1] Informatics Institute, University of Amsterdam, The Netherlands
mark.voschezang@icloud.com
[2] Nikhef, The Netherlands martinfr@nikhef.nl

Abstract. Holographic projections are volumetric projections that make use of the wave-like nature of light and may find use in applications such as volumetric displays, 3D printing, lithography and LIDAR. Modelling different types of holographic projectors is straightforward but challenging due to the large number of samples that are required. Although computing capabilities have improved, recent simulations still have to make trade-offs between accuracy, performance and level of generalization. Our research focuses on the development of optimizations that make optimal use of modern hardware, allowing larger and higher-quality simulations to be run. Several algorithms are proposed; (1) a brute force algorithm that can reach 20% of the theoretical peak performance and reached a 43× speedup w.r.t. a previous GPU implementation and (2) a Monte Carlo algorithm that is another magnitude faster but has a lower accuracy. These implementations help researchers to develop and test new holographic devices.

Keywords: Computer-Generated Holograms (CGH) · Digital Holography · GPU Computing (GPGPU) · CUDA · Monte Carlo Integration

1 Introduction

Holography was invented as early as 1947 by Dennis Gabor, but high-resolution holographic projectors are still not commercially viable[9]. Holograms are the product of the interference of light-waves[10]. They may find use in applications such as volumetric displays, 3D printing or LIDAR. Figure 1 shows examples of two holographic projections.

Modern holographic projectors give full control over an array of light sources and can display arbitrary distributions of light. The "pixels" of such a projector emit light of a specific wavelength and can be tuned in intensity and phase. A major limitation of projectors is the limited pixel pitch of projectors. Due to the small wavelength of visible light (0.4 μm to 0.7 μm), holographic projectors would need a pixel pitch of less than 0.2 μm in order to correctly reproduce the correspondingly small structures in a light field. A larger pixel pitch will cause under-sampling artefacts in the projections. Typical under-sampling effects can

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77970-2_11

Fig. 1: Two holographic projections; a tilted ring (left) and a question mark symbol (right). The diamond shaped spot is the light source of the projector.

be reduced by placing the pixels at a-periodic or random intervals[13,21]. In order to study these effects, a prototype of a holographic projector is being developed, shown in Figure 2. The role of simulations is to predict the effect that certain components will have on the projections (e.g. the size and positions of apertures).

The behaviour of light is a common subject to study and there are various models that can be used under different circumstances[10]. This paper relies on a single, generic model, namely a point-source model of the electric field component of light. The projector consists of a number of discrete point light sources of which the resulting light field is calculated at discrete points in the projection volume. The light intensity distribution at various positions is obtained by superimposing the received light components at those positions. Such a component can be described by

$$E(t) = \frac{A}{\delta} \exp\left(i\left(\phi + \frac{2\pi\delta}{\lambda} + \omega t\right)\right),$$

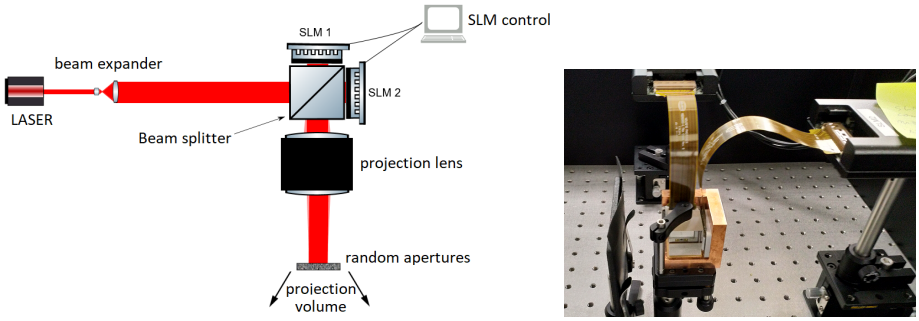


Fig. 2: A schematic (left) and a picture (right) of an experimental setup of a holographic projector. A beam splitter first splits the light emitted from a laser into two beams that are reflected by the Spatial Light Modulators (SLM 1 and 2). The reflected light is then superimposed at the beam splitter and directed through a projection lens and a mask that contains apertures at specific positions.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77970-2_11

where δ is the distance from the source, A is the source strength, ϕ the phase offset of the light source, λ is the wavelength and ω is the frequency of the light. Because the time-dependent component is the same everywhere in the projection volume it can be omitted; only the phase offset of the source ϕ and the phase shift over distance $\frac{2\pi\delta}{\lambda}$, are taken into account. The superposition of these components can be formulated as either a summation or an integral (w.r.t space). The result can be expressed as a *phasor* or as a polar coordinate. The computational complexity is quadratic w.r.t the number of sample points and the computations have to be done in double precision to prevent significant rounding errors. The spatial frequency of two beams under an angle α is is given by $\frac{2\sin\frac{\alpha}{2}}{\lambda}$. Simulating large volumes using this approach with at least two samples per wavelength requires an enormous amount of computing power.

Recently, multiple optimizations for holographic simulations for have been developed for modern hardware[17,23]. The majority of these models use assumptions that reduce the computational complexity of the problem. An example of this is the Fresnel approximation, which assumes that sample points lie in the near field[10,27]. Compressive holography is a more complex method that uses compressive sensing to approximate the projection distribution with a relatively low number of sample points[6,26,28]. These models are applicable to a limited number of projector configurations. As a result, these models cannot be used to study arbitrary projector designs. Hence our choice of model makes a qualitative difference.

Two algorithms, that first optimize computational efficiency and then complexity, have been developed. The corresponding implementations will aid physicists to run larger and more detailed simulations, which they can use to develop and test new holographic devices.

2 Background

This section gives a brief overview of relevant background material.

2.1 GPU Computing

Over the past decades, the performance and capabilities of computing hardware has increased[3]. Whereas CPU's are generally optimized to have a low latency, for example by maximizing clock frequency, GPU's are designed to do a narrow scope of tasks very efficiently[15]. They achieve this by making use of massive parallelism.

The extend to which applications can make optimal use of GPU hardware differs. A standard metric to compare performance is throughput, defined as the number of FLoating-point OPerations per Second (FLOPS). State-of-the-art linear algebra algorithms can reach 61.4%, 86.8% and over 90% of the theoretical peak performance for various GPU architectures[25,1,12]. In case of smaller, sub-optimal input sizes the efficiency decreases, for example to 30%[2]. Depending on the sparsity of the input, state-of-the-art sparse matrix algorithms may only

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77970-2_11

reach a fraction (< 0.08 %) of the the theoretical peak performance[19]. These results shows that making optimal use of GPU hardware can be difficult, even for fundamental applications.

This research focuses on NVIDIA GPU's and the CUDA Programing Model. CUDA revolves around *kernels* which are Single Instruction Multiple Thread (SIMT) functions[7]. In other words, a single function is performed in parallel on many different threads or cores, with different input values per thread. Kernels are executed on a *grid* (representing the whole GPU), which contains *blocks* of threads (representing multiprocessors and cores). These abstractions allow arbitrary grid and block sizes to be used for different kernels, independent from the GPU that the program will run on. The number of grids, blocks and threads can be higher than the actual number of corresponding hardware units. Schedulers on the GPU act as multiplexers, making use of interleaved execution of tasks to maximize the utilization of hardware. In addition, the parallel execution of kernels and data transfers in CUDA can be managed using *streams*. Streams can be described as independent virtual containers that allow data transfers and kernel executions to happen in parallel.

2.2 Numerical Integration

Integrals can be approximated using two general techniques that are based on summations[8]. The first technique discretizes space and sums the resulting components, for example using a middle Riemann sum. The main disadvantage of this method is that the granularity of the discretization may cause systematic errors. The second technique is called Monte Carlo sampling and uses randomly distributed sample points[22]. For a D-dimensional unit hypercube it has the form

$$\int f(\mathbf{x})\, d\mathbf{x} \approx \frac{1}{N} \sum_{n=1}^{N} f(\mathbf{u}_n) \qquad \mathbf{u}_n \sim \mathcal{U}(0,1)^D.$$

The summation converges, with high probability, for large enough N. This means that it is possible to make a tradeoff between accuracy and computational cost by using fewer sample points.

Variance Reduction. It can be shown that the mean absolute error of an MC estimate grows with a factor $\frac{\sigma}{\sqrt{N}}$, where σ is the standard deviation of the estimator[16]. This means that the convergence of MC estimates can be improved by constructing an estimator with lower variance. There exist a variety of methods that achieve this, but they usually rely on a priori knowledge about the underlying distribution. This paper focuses on general solutions and no prior knowledge is assumed. Therefore the technique is stratified sampling is chosen[22]. It uses a conditional variable to spread out the samples over the sample space, making each sample set more representative of the original distribution.

An alternative variance reduction method is importance sampling[22]. This approach focuses the sampling on the areas that contribute the most to the

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77970-2_11

estimation. This technique has successfully been applied to holography, albeit for an older generation of hardware[5].

3 Methods

This section introduces a number of algorithms that compute superpositions, gradually increasing in complexity. **Kernels** compute *partial* superpositions for multiple source datapoints at multiple target positions. They are combined with **estimators** that use these kernels to make estimations of *full* superpositions.

A number of CUDA-variables are are used. Threads and blocks are indexed with a *threadIdx* and a *blockIdx* and can be divided in multiple dimensions (x, y, z). The number of threads per block is denoted by *blockDim* and the number of blocks per grid is denoted by *gridDim*. For convenience we define *gridSize* as the total number of threads per grid.

3.1 Superposition Kernels

All superposition kernels compute the partial superpositions of a set of source phasors (with three-dimensional positions $\mathbf{u}$) at certain target positions $\mathbf{v}$. They use a SIMT approach where each GPU thread applies the following function to a subset of the input data:

$$f(A, \phi, \mathbf{u}, \mathbf{v}) := \frac{A}{\delta} \exp\left(i\left(\phi \pm \frac{2\pi\delta}{\lambda}\right)\right) \qquad \delta \equiv |\mathbf{u} - \mathbf{v}|. \tag{1}$$

These partial results are written to a matrix $\mathbf{Y}$. Note that the matrix $\mathbf{Y}$ may not fit in GPU memory; this problem is solved in the next section by splitting the dataset up in chunks. After a superpositon kernel has terminated a second kernel is used sum the rows of this matrix, resulting in a vector that contains the full superpositions. This second kernel is a simple matrix-vector product, for which we have used the library function *cublasZgemv*[18].

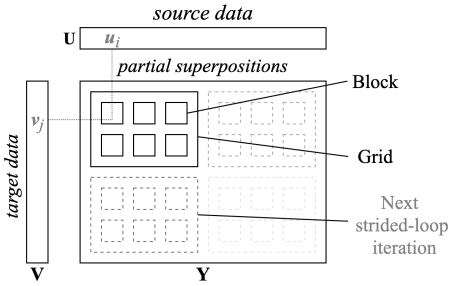


Fig. 3: The data-structures used in the superposition kernels.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77970-2_11

Naive Kernel. The first thread dimension (x) is mapped to the source data and the second dimension (y) is mapped to the target data. The Naive kernel iterates over the source and target datapoints using a *strided* loop, allowing the kernel to be used for arbitrary input sizes. This approach is shown graphically in Figure 3 and in pseudocode in Algorithm 1.

Algorithm 1: Naive Superposition Kernel

Kernel Superposition($\mathbf{a}, \boldsymbol{\phi}, \mathbf{U}, \mathbf{V}, \mathbf{Y}$) :
Result: Written to $y_{m,n} \in \mathbf{Y} \in \mathbb{C}^{M \times N}$
input : source phasors with polar coordinates $\mathbf{a}, \boldsymbol{\phi} \in \mathbb{R}^N$,
source positions $\mathbf{u}_n \in \mathbf{U}$ for $n = 1, \ldots, N$,
target positions $\mathbf{v}_m \in \mathbf{V}$ for $m = 1, \ldots, M$
$\langle t_x, t_y \rangle \leftarrow$ GlobalThreadIdx();
for $n \leftarrow t_x; \quad n \leq N; \quad n \leftarrow n + gridSize.x$ **do**
  // $a_n, \phi_n, \mathbf{u}_n$ can be cached here
  for $m \leftarrow t_y; \quad m \leq M; \quad m \leftarrow m + gridSize.y$ **do**
    $y_{m,n} \leftarrow f(a_n, \phi_n, \mathbf{u}_n, \mathbf{v}_m)$; // Equation 1

Advanced Kernels. The memory complexity for reading and writing is $\mathcal{O}(NM)$ for the naive kernel (excluding any additional caching). The memory complexity of the subsequent summation kernel is $\mathcal{O}(NM)$ as well because it uses the output of the superposition kernel as input. We will consider two optimizations that reduce the writing frequency and refer to them as the Reduced kernel and the Shared kernel. They are formalized together in Algorithm 2, where the Boolean *shared_memory* is used to distinguish the two approaches.

Algorithm 2: Reduced & Shared Superposition Kernel

Kernel Superposition($\mathbf{a}, \boldsymbol{\phi}, \mathbf{U}, \mathbf{V}, \mathbf{Y}$) :
Result: Written to $y_{m,n} \in \mathbf{Y}$
$\langle t_x, t_y \rangle \leftarrow$ GlobalThreadIdx();
for $m \leftarrow t_y; \quad m \leq M; \quad m \leftarrow m + gridSize.y$ **do**
  if $t_x > N$ **then** continue to next iteration;
  $y' \leftarrow 0$; // local temporary memory
  for $n \leftarrow t_x; \quad n \leq N; \quad n \leftarrow n + gridSize.x$ **do**
    $y' \leftarrow y' + f(a_n, \phi_n, \mathbf{u}_n, \mathbf{v}_m)$;
  if *shared_memory* **then**
    $y' \leftarrow$ block-level reduction of y'; // using shared memory
    if $t_y = 1$ **then** $y_{m,1+\text{blockIdx.x}} \leftarrow y'$;
  else
    $y_{m,t_x} \leftarrow y'$;

The *Reduced* kernel aggregates partial superposition results locally, at the thread-level. This reduces the writing frequency. The inner and outer loops are switched s.t. the inner loop becomes a direct summation of partial superposi-

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77970-2_11

tions. As a result, writing to global memory happens only once per summation and there is no additional local memory required.

The *Shared* kernel reduces the writing frequency further by using shared data to aggregate the partial results of all threads in a block. The amount of block-level reduction is a trade-off between workload per thread, synchronization per block and global memory access. The library CUB provides a number of reductions that can be used inside CUDA kernels [14,11].

3.2 Estimators

The partial superpositions computed by the kernels are combined by an estimator. The estimators split the dataset up and send them to the GPU in batches, using different (concurrent) streams, as shown in Figure 4. The **deterministic estimator** does this sequentially and is straightforward to implement. The stochastic estimators are more complex and are given below. This paper describes three variants. They all use a discrete source dataset, but they can easily be transformed to the case of a continuous source distribution.

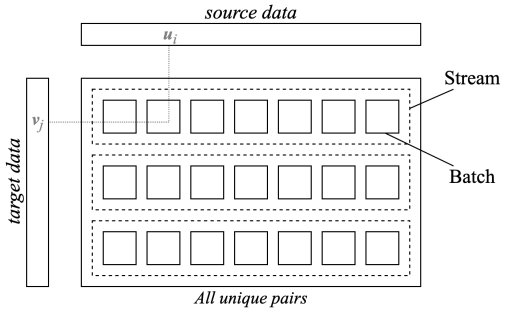


Fig. 4: Data traversal using batches and streams. Each batch corresponds to a single kernel, as shown in Figure 3. Multiple streams can be active at the same time but the batches within a stream are executed in series (from left to right).

The **stochastic estimators** improve performance by using Monte Carlo (MC) sampling to reduce the number of source datapoints that are used. A trivial solution would be to use the deterministic estimator with a random subset of the source dataset. However, this would increase the risk of overfitting (overgeneralization) because all target datapoints would use the same subset of data. The purpose of the stochastic estimators is enforce that target datapoints can sample independently from the source dataset.

Figure 5 shows a simplified representation of a stochastic estimator for a single stream. First the source data is shuffled and distributed over the available streams. Each stream independently computes the partial superpositions of multiple batches using either one of the kernels described in the previous section. This is repeated until a stream has seen enough source datapoints. The

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77970-2_11

partial results are then summed and checked for convergence. The stream is terminated when either the corresponding superpositions have converged or when the maximum number of iterations has been reached.

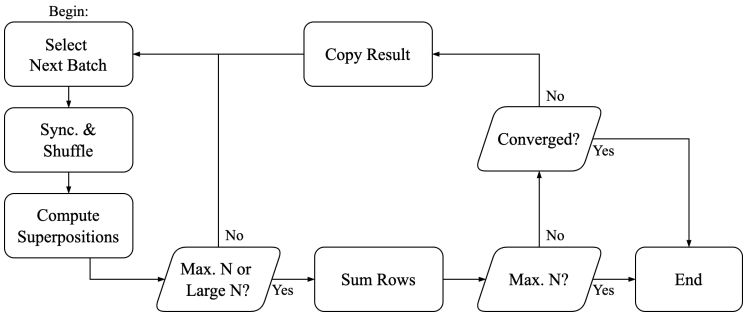


Fig. 5: A flowchart of the Stochastic Estimator, for a single stream. This scheme is repeated until all batches are completed. The step "Sync. & Shuffle" is optional and can be omitted if the target datapoint are chosen during kernel execution.

Sampling Methods The general stochastic estimator algorithm can make use of two different sampling methods. We will refer to these as True MC and Batched MC. Both of these can be implemented using either conditional and unconditional sampling. The relevant combinations are visualized in Figure 6.

The *True MC* estimator adheres to the traditional MC method. Instead of iterating over a range of indices, the indices are chosen randomly inside the superposition kernel. The disadvantage of this method is that it results in random access of global memory, which is inefficient.

The *Batched MC* estimator is slightly more complex. Datapoints are grouped together s.t. each kernel accesses a pre-specified subset of data. This avoids the

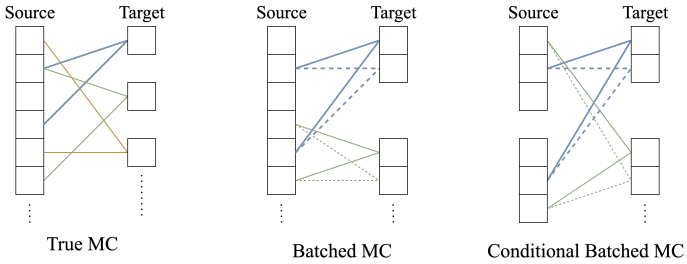


Fig. 6: Monte Carlo estimators. The lines indicate which source datapoints are used by which target datapoints. For Batched MC the *target* datapoints are grouped and for conditional MC the *source* datapoints are grouped as well.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77970-2_11

random access pattern inside the superposition kernel, without increasing the memory usage. The available input dataset is distributed between the streams and then shuffled. This independent reshuffling means that the sampling is done with replacement. The main issue with this approach is that the target sample points within batches become correlated due to their shared input sample set.

The *Conditional Batched MC* estimator mitigates this effect by using conditional sampling. This forces the samples to become more representative of the whole dataset.

Convergence. The stochastic estimators combine two relatively simple convergence criteria, which are both based on a current estimate and a previous estimate. The first criteria computes the absolute difference between the (normalized) amplitudes of the two estimations and compares it to a threshold. The second criteria makes use of the fact that a sum of random vectors grows with the square root of the number of elements in that sum[3]. It is assumed that certain, far-away phasors are distributed as random vectors. The second convergence criteria normalizes the amplitudes of the estimations by dividing them by the square root of the sample sizes.

4 Experiments & Results

This section contains a number of experiments that test the performance of the kernels and estimators. The three kernels are abbreviated as **#1**: Naive kernel, **#2**: Reduced kernel (with thread-level aggregation) and **#3**: Shared kernel (with block-level aggregation using shared data). The estimators are the deterministic estimator, which uses the full dataset, and the three stochastic estimators, which use random subsets of data. The implementations are available online[24].

The experiments are run on a machine with an Intel Xeon E5-1650 CPU and a NVIDIA Quadro GV100 GPU (generation Volta), with a peak performance of 7.4 TFLOPS (double precision). The level of compiler-optimization is set to default (-O3).

The following metrics are used. *Runtime* is measured after initialization of all CPU datastructures but before initialization of the GPU datastructures. *Speedup* is defined as the mean runtime of a baseline implementation divided by the runtime of an optimized implementation. *Efficiency* or throughput is is measured in double precision FLOPS [4]. The total number of floating-point operations for

[3] A sum of N random vectors can be represented by a two-dimensional random walk of N steps. The corresponding distribution is derived by (and named after) Rayleigh and is given by $p(\ell) = \frac{2\ell}{N} \exp\left(-\ell^2/N\right)$, where ℓ is the length or absolute value of the phasor-sum[20]. The expected value of this distribution is proportional to $\sqrt{N}$. This distribution requires the input phasors to have unit length, but more refined solutions that allow for arbitrary amplitude and phase distributions exist as well[4].

[4] The exact number of floating point operations is difficult to determine because compilers can restructure code and certain implementations are hardware-dependent.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77970-2_11

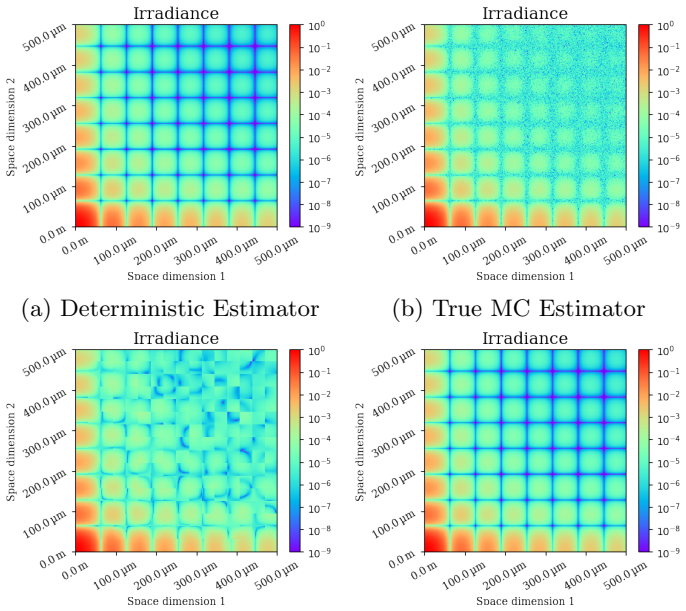
the superpositions of N source datapoints and M target datapoints is defined as $29NM$.

The validation of the model and implementations is done using a prototype of a holographic projector. These experiments are important but are not discussed here.

4.1 Accuracy

The accuracy of the stochastic estimators depends on the input values and parameters such as convergence threshold and batch size. A single typical input distribution is used to show the qualitative differences that can be caused by the different estimators. The irradiance of the resulting projections is shown in Figure 7. The projection generated by the deterministic estimator projection is considered to be the ground truth. Overall, the stochastic estimators produce

For example fused multiply-add operations and hardware-units for special arithmetic such as the square root[7].



(a) Deterministic Estimator

(b) True MC Estimator

(c) Batched MC Estimator

(d) Conditional Batched MC Estimator

Fig. 7: Projections generated by four different estimators, shown with a logarithmic color-scale. The projection plane is parallel to a projector with 512× 512 pixels (at a distance of 35 cm) and limited to the top-right quadrant w.r.t the center of the projector screen. The projection is sampled using 1024×1024 sample points and the convergence threshold is $\epsilon = 10^{-4}$,

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77970-2_11

accurate results in high amplitude regions, but start to diverge in low-amplitude regions. The True MC projection contains uniform-like noise. The unconditional Batched MC projection shows intra-batch correlations that are not present in the ground truth. The Conditional Batched MC projection is the most accurate, but still contains differences with the ground truth.

4.2 Kernel Performance

Figure 8 shows the performance of the three kernels as function of the number of source datapoints. A linear least-squares regression model is fitted and was significant with a p-value of 0.001. Kernels #2 and #3 clearly outperform kernel #1. The difference between kernels #2 and #3 is small, but kernel #2 does have a lower slope.

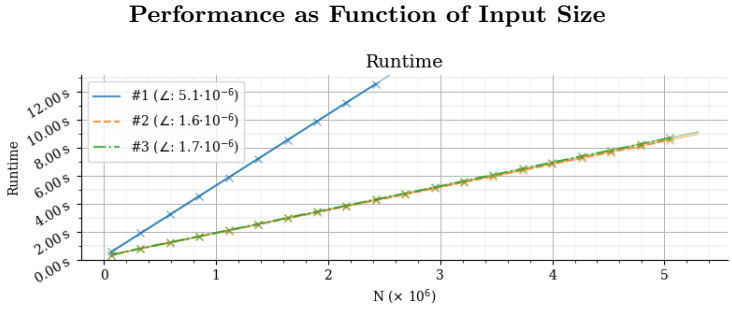


Fig. 8: The performance of the Naive Kernel (#1), the Reduced Kernel (#2) and the Shared Kernel (#3). The slope of linear fit is denoted by $\angle$. The parameters are: $M = 256^2$, gridDim = blockDim = $\langle 8, 8 \rangle$, thread size = $\langle 64, 8 \rangle$.

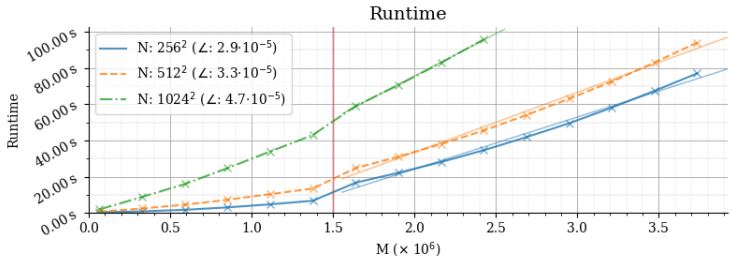


Fig. 9: Performance as function of output size for the Reduced Kernel (#2). The slope of linear fit using the data after the red vertical line is denoted by $\angle$. The parameters are: gridDim = blockDim = $\langle 8, 8 \rangle$, thread size = $\langle 256, 16 \rangle$.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77970-2_11

Figure 9 shows the performance of kernel #2 as function of the number of target datapoints. In contrast to the previous results, the performance is no longer linear. The runtime increases monotonically, but becomes slightly convex after $1.5 \cdot 10^6$ target datapoints. However, the independence of the target dataset allows the computations to be split up into independent chunks with a size that maximizes the efficiency per chunk. Using this technique would still result in linear computational growth.

An existing implementation, written in Matlab, is used as a baseline. It runs on the same hardware but uses single-precision data, which gives it an performance advantage. Figure 10 shows the efficiency and speedup of the three algorithms in comparison with the Matlab implementation. The Reduced kernel has the best performance, with a speedup of 43.9, followed by the Shared kernel with a speedup of 41.1. The efficiency of the Reduced is 15.3 TFLOPS, which is 20.7% of the theoretical peak performance of this GPU. The rest of the experiments will use the kernel #2.

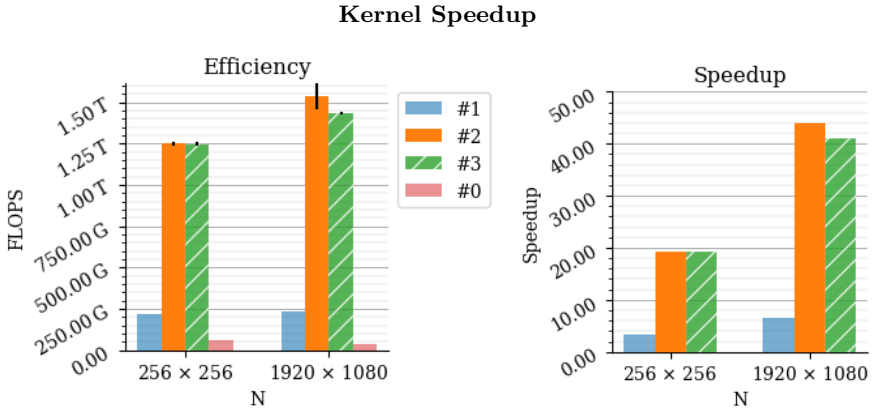


Fig. 10: The performance of the three CUDA kernels in comparison with the Matlab implementation (labelled as #0). The right graph does not contain error bars. The shared parameters are $M = 1024 \times 1024$ and gridDim $= \langle 16, 16 \rangle$. For algorithm #1 the remaining parameters are thread size $= \langle 32, 32 \rangle$, blockDim $= \langle 8, 8 \rangle$ and for algorithm #2 and #3 the parameters are thread size $= \langle 512, 256 \rangle$, blockDim $= \langle 16, 16 \rangle$.

4.3 Stochastic Estimator Performance

Figure 11 shows the performance of the unconditional estimators for different convergence thresholds. Batched MC is roughly an order of magnitude faster than True MC. The lowest threshold (10^{-16}) is used to prevent convergence and thus represent the worst-case scenario. The speedups for a convergence threshold of 0.01 w.r.t a non-converging result is 15.3 for True MC and 10.6 for Batched MC. For larger input sizes the relative speedups should be even higher.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77970-2_11

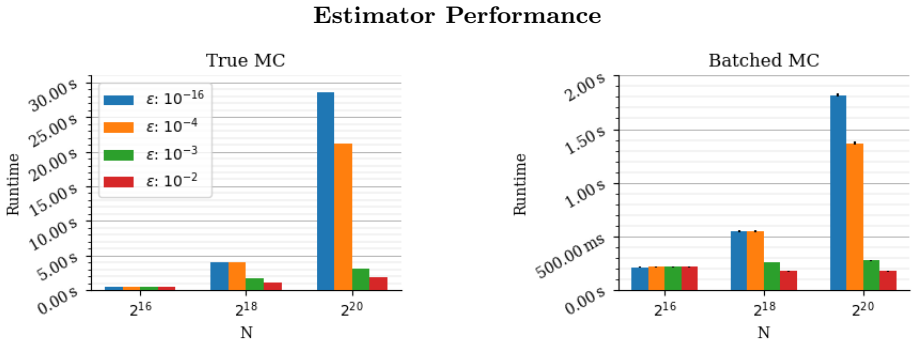


Fig. 11: Comparison of unconditional MC estimators for different input sizes and convergence thresholds. The parameters are: $M = 256^2$, gridDim $= \langle 8, 8 \rangle$, blockDim $= \langle 16, 16 \rangle$,

5 Conclusion

This research proposes two optimizations of simulations of holographic projectors. Because the underlying model does not rely on complex assumptions it can be used to model arbitrary projectors. The proposed (double-precision) implementation reaches a speedup of a factor 43.9 compared to a previous (single-precision) implementation, using the same hardware. Moreover, it reached over 20% of the theoretical peak performance, which compares favorably to state-of-the-art GPU implementations in other domains. This also shows that additional speedups that are larger than a factor five cannot be reached with the same hardware (and the same mathematical model).

A second implementation circumvents this hardware-related upper bound by reducing the number of operations per estimation. This is achieved using Monte Carlo integration. Although this comes with a decrease in accuracy, it can be used to give rough estimates. The results can subsequently be verified using the first implementation.

References

1. Abdelfattah, A., Baboulin, M., et al.: High-performance tensor contractions for gpus. Procedia Computer Science **80**, 108–118 (2016)
2. Abdelfattah, A., Haidar, A., et al.: Performance, design, and autotuning of batched gemm for gpus. In: International Conference on High Performance Computing. pp. 21–38. Springer (2016)
3. Asanovic, K., Bodik, R., Catanzaro, B.C., et al.: The landscape of parallel computing research: A view from berkeley. Tech. Rep. UCB/EECS-2006-183, EECS Department, University of California, Berkeley (Dec 2006)
4. Beckmann, P.: Statistical distribution of the amplitude and phase of a multiply scattered field. Journal of Research of the National Bureau of Standards, 66D **3**, 231–240 (1962)

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77970-2_11

5. Bokor, N., Papp, Z.: Monte carlo method in computer holography. Optical Engineering **36**, 1014–1020 (1997)
6. Brady, D.J., Choi, K., Marks, D.L., Horisaki, R., Lim, S.: Compressive holography. Optics express **17**(15), 13040–13049 (2009)
7. Corporation, N.: CUDA C++ Programming Guide (October 2020), version 11.1.1
8. Davis, P.J., Rabinowitz, P.: Methods of numerical integration. Courier Corporation (2007)
9. Gabor, D.: A new microscopic principle. nature **161**, 777–778 (1948)
10. Hecht, E., Zajac, A.: Optics. Addison Wesley (1974)
11. Kogge, P.M., Stone, H.S.: A parallel algorithm for the efficient solution of a general class of recurrence equations. IEEE transactions on computers **100**(8), 786–793 (1973)
12. Li, X., Liang, Y., Yan, S., et al.: A coordinated tiling and batching framework for efficient gemm on gpus. In: Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming. pp. 229–241 (2019)
13. Li, Z.: Principle and characteristics of 3d display based on random source constructive interference. Optics express **22**(14), 16863–16875 (2014)
14. Merrill, D.: Cub documentation (2020), accessed: 2020-08-01
15. Nickolls, J., Dally, W.J.: The gpu computing era. IEEE micro **30**(2), 56–69 (2010)
16. Niederreiter, H.: Random number generation and quasi-Monte Carlo methods. SIAM (1992)
17. Nishitsuji, T., Shimobaba, T., et al.: Review of fast calculation techniques for computer-generated holograms with the point-light-source-based model. IEEE Transactions on Industrial Informatics **13**(5), 2447–2454 (2017)
18. NVIDIA Corporation: The API Reference guide for cuBLAS (v11.0.3) (2020)
19. Pal, S., Beaumont, J., Park, et al.: Outerspace: An outer product based sparse matrix multiplication accelerator. In: 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA). pp. 724–736. IEEE (2018)
20. Rayleigh, J.W.S.B.: The theory of sound, vol. 2. Macmillan (1896)
21. Rivenson, Y., Stern, A., Javidi, B.: Compressive fresnel holography. Journal of Display Technology **6**(10), 506–509 (2010)
22. Ross, S.M.: Simulation. Academic Press (2012)
23. Tsang, P., Poon, T.C., Wu, Y.: Review of fast methods for point-based computer-generated holography. Photonics Research **6**(9), 837–846 (2018)
24. Voschezang, M.: Holographic Projector Simulations (2020), `github.com/voschezang/Holographic-Projector-Simulations/tree/snapshot-stochastic-estimators`
25. Younge, A.J., Walters, J.P., Crago, S., Fox, G.C.: Evaluating gpu passthrough in xen for high performance cloud computing. In: 2014 IEEE International Parallel & Distributed Processing Symposium Workshops. pp. 852–859. IEEE (2014)
26. Zhang, H., Cao, L., Zhang, H., Zhang, W., Jin, G., Brady, D.J.: Efficient block-wise algorithm for compressive holography. Optics express **25**(21), 24991–25003 (2017)
27. Zhao, T., et al.: Accelerating computation of cgh using symmetric compressed look-up-table in color holographic display. Optics express **26**(13), 16063–16073 (2018)
28. Zhao, Y., Cao, L., et al.: Accurate calculation of computer-generated holograms using angular-spectrum layer-oriented method. Optics express **23**(20), 25440–25449 (2015)

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77970-2_11