# The CUDA LATCH Binary Descriptor: Because Sometimes Faster Means Better

Christopher Parker[1], Matthew Daiter[2], Kareem Omar[3], Gil Levi[4] and Tal Hassner[5,6]

[1]University of Oslo, Norway [2]Nomoko AG
[3]University of Alabama in Huntsville, AL, USA
[4]Tel Aviv University, Israel
[5]Information Sciences Institute, USC, CA, USA
[6]The Open University of Israel, Israel

**Abstract.** Accuracy, descriptor size, and the time required for extraction and matching are all important factors when selecting local image descriptors. To optimize over all these requirements, this paper presents a CUDA port for the recent Learned Arrangement of Three Patches (LATCH) binary descriptors to the GPU platform. The design of LATCH makes it well suited for GPU processing. Owing to its small size and binary nature, the GPU can further be used to efficiently match LATCH features. Taken together, this leads to breakneck descriptor extraction and matching speeds. We evaluate the trade off between these speeds and the quality of results in a feature matching intensive application. To this end, we use our proposed CUDA LATCH (CLATCH) to recover structure from motion (SfM), comparing 3D reconstructions and speed using different representations. Our results show that CLATCH provides high quality 3D reconstructions at fractions of the time required by other representations, with little, if any, loss of reconstruction quality.

## 1 Introduction

*Quantity has a quality all its own*
Thomas A. Callaghan Jr.

Local features and their descriptors play pivotal roles in many computer vision systems. As such, research on improving these methods has been immense. Over the years, this effort yielded progressively more accurate representations. These improvements were often demonstrated on standard benchmarks designed to measure the accuracy of descriptor matching in the presence of various image transformations and other confounding factors. It remains unclear, however, if the improved accuracy reported on these benchmarks reflects better, more useful representations when used in real world computer vision systems.

Take, for example, recent attempts to use deep learning for image feature representation (e.g., [10,26,40]). There is no question that given sufficient training data and computational resources deep learning methods can achieve astonishing accuracy. Hence, using them to obtain local descriptors can result in better representations and by so doing impact a wide range of computer vision systems.

But using deep learning for feature description and matching does not come without a price: Most of these methods are computationally expensive and even with graphical processing units (GPU), are relatively slow. Even after extraction, their dimensions and floating point values makes them slow to match. Finally, they require substantial training data which can be difficult to provide.

These limitations should be contrasted with evidence that accuracy, though important, is not the only property worth considering when choosing descriptors. For example, simultaneous localization and mapping (SLAM) methods were shown to obtain better 3D reconstructions for the same computational effort by increasing the amount of feature points per keyframe [29]. This suggests that computationally cheaper features are more desirable for these systems. Consequently, state of the art SLAM techniques [21] use ORB [25] rather than more accurate but computationally expensive representations such as SIFT [19]: Doing so allows for a greater number of features to be extracted without compromising reconstruction accuracy. In fact, even classification systems appear to benefit from having more features over higher feature accuracy, as reported by [22].

One side effect to the success of deep learning is that the hardware enabling it – GPU processors – is now becoming standard on systems running computer vision applications, including even consumer cellphone devices. Beyond deep learning, these GPUs can also be used to accelerate extraction and matching of older, so-called *engineered* descriptors. These representations may not reach the same benchmark performances as deep learning techniques, but their extraction on the GPU offers a potential trade off between accuracy and run time. In particular, faster descriptor extraction and matching allows for more descriptors to be used and consequently better overall system performances.

GPU accelerated features were considered in the past. We, however, focus on a particular binary descriptor: the *Learned Arrangement of Three Patches* (LATCH) [18]. It was recently shown to offer a compromise between the high accuracy, low speeds of floating point representations such as SIFT [19], and the low accuracy, high speeds of binary descriptors (e.g., ORB [25], BRIEF [8]). Beyond these properties, its design also happens to neatly fit GPU processing.

Our contributions are: **(1)** We describe CLATCH, a CUDA port for LATCH, enabling descriptor extraction and matching directly on the GPU. **(2)** We embed CLATCH in the OpenMVG library [20], along with a fast, GPU based Hamming distance, brute force descriptor matcher. Finally, **(3)** we compare SfM 3D reconstructions on scenes from [23] using SIFT, recent deep learning based representations and our CLATCH. These show that CLATCH reconstructions are comparable or even better than those obtained with other representations, yet CLATCH requires a fraction of the run time of its alternatives. Importantly, to promote reproducibility, the code used in this paper is publicly available from the project webpage: `www.openu.ac.il/home/hassner/projects/LATCH`.

## 2 Related work

Due to their key role in many computer vision systems, local feature descriptors are extensively studied. A comprehensive survey is therefore outside the scope of this paper. Below we provide only a cursory overview of this topic.

**Floating point representations.** For nearly two decades now, SIFT [19] is very likely the most widely used local image descriptor. It and the representations that followed (e.g., SURF [6]) represent the region around an image pixel using a vector of typically 128 floating point values. This vector is often a histogram of measurements extracted from the image, most commonly various functions of the local intensity gradients.

**Binary descriptors.** Despite the success of the older floating point representations, a prevailing problem was their extraction time and dimensionality (which, in turn, affected their storage and matching time). In response, binary descriptors were proposed as low dimensional, efficient alternative representations. These typically assign descriptor values by quick, pixel intensity comparisons.

One of the first binary descriptors was the Binary Robust Independent Elementary Features (BRIEF) [8], soon followed by the Oriented fast and Rotated BRIEF (ORB) descriptor [25] which added rotation invariance, the Binary Robust Invariant Scalable Keypoints (BRISK) [17] which used a more effective pixel sampling pattern, and the Fast REtinA Keypoint descriptor (FREAK) [1] which sampled intensities using a pattern similar to the one found in human retinas. The Accelerated-KAZE (A-KAZE) was suggested in [2]. It builds on the earlier Local Difference Binary (LDB) descriptor [38,39] by computing the binary descriptor values from mean image intensities over a range of patch sizes. The binary online learned descriptor (BOLD) [4] improve accuracy yet retain high processing speeds. Finally and very recently, the LATCH binary descriptors were proposed in [18]. We defer discussion of LATCH to Sec. 3.

Hybrid binary/floating-point methods were also suggested. One example is LDA-Hash [30] which extracts SIFTs, projects them to a discriminative space and applies a threshold to obtain binary descriptors. DBRIEF [34] instead uses patch intensities directly, BinBoost [16,33] learns a set of hash functions corresponding to each bit in the final descriptor and PR-proj [27] uses learning and dimensionality reduction to produce compact binary representations. The computational effort required to extract these descriptors is similar to (if not greater than) floating point descriptors. The representations, however, are short binary vectors and so matching and storing them is relatively efficient.

**Computing local descriptors on the GPU.** Of course, we are not the first to propose porting local feature extraction to the GPU. To our knowledge, nearly all these efforts used the GPU to aid in extracting *floating point descriptors*, including GPU-SIFT (see, e.g., [37,13,28,36]) and GPU-SURF [32]. These methods all used GPUs in portions of the extraction process. For example, [36] used the GPU only to compute convolutions, all other stages performed on the CPU. In addition, and more importantly, the gain in performance reported by these methods are modest and do not approach the speeds of contemporary binary descriptors, let alone our CLATCH.

Interestingly, the only available GPU *binary descriptor* is CUDA ORB, implemented by OpenCV [14]. As we later discuss, due to the nature of GPU processing, the run time advantage of ORB over the more accurate LATCH descriptor when computed on the CPU, vanishes on the GPU.

**Deep features.** Following the remarkable success of deep learning in computer vision, it is no surprise that these methods are also being applied to feature point description. Convolutional Neural Networks (CNN) were used in a number of previous attempts to learn local descriptor representations [10,26,40,3].

In most cases, a Siamese deep network is trained with hinge loss [10,26,40]. The training set used in these cases consists of positive and negative labeled patch pairs. Metric learning is then used to improve matching. Finally, [3] proposed an efficient CNN design, bringing processing speeds down substantially. As we later show, their run time is still slower than our proposed approach.

## 3 CUDA LATCH

### 3.1 Preliminaries

**The LATCH feature descriptor.** LATCH was recently introduced in [18] and is available as part of the OpenCV library since ver. 3.0 [14]. Its design was inspired by the observation that *pure* binary descriptors such as BRIEF and ORB produce their values by comparing pairs of pixel intensities, a process which can be sensitive to local noise. To address this, these methods used various smoothing techniques before pixel values were compared. Smoothing, however, has the adverse effect of losing important high frequency image information.

Rather than smoothing the image and then comparing single pixel values, LATCH computes its binary values by comparing pixel *patches*. The LATCH descriptor for image pixel $\mathbf{p} = (x, y)$ is computed by selecting $t = 1..T$ patch triplets, one for each LATCH bit. For triplet $t$, three pixels are selected in the region around $\mathbf{p}$: an *anchor* pixel $\mathbf{p}_{t,A}$ and two *companion* pixels $\mathbf{p}_{t,1}$ and $\mathbf{p}_{t,2}$. The $k \times k$ pixel patches, $\mathbf{P}_{t,A}, \mathbf{P}_{t,1}$, and $\mathbf{P}_{t,2}$ centered on each of these three pixels are extracted. Finally, bit $t$ in the LATCH descriptor for $\mathbf{p}$ is set by comparing the Frobenious norm of the anchor to its two neighbors, as follows:

$$LATCH(\mathbf{p}, t) = \begin{cases} 1 & \text{if } ||\mathbf{P}_{t,A} - \mathbf{P}_{t,1}||_F^2 > ||\mathbf{P}_{t,A} - \mathbf{P}_{t,2}||_F^2 \\ 0 & \text{otherwise} \end{cases} . \quad (1)$$

The triplets LATCH uses are fixed but are not arbitrary: Triplets are selected during training using the data set from [7], which contains same/not-same labeled image windows. Triplets were chosen by considering how well their bits correctly predicted the same/not-same labels over the entire training set. To prevent choosing correlated triplets, following [1,25], triplets are skipped if their predictions are correlated with those of previously chosen triplets.

In their work [18], LATCH contained 512 bits (selected triplets) each one representing triplets of $7 \times 7$ patches. At matching time, its computational requirements were obviously equal to those of any other 512 bit binary descriptor.

Due to the use of patches and multiple Frobenious norms, extracting LATCH was slower than pure binary descriptors of the same size. Experiments reported in [18], however, showed that the increase in extraction time was small. This was balanced by improved accuracy which bested existing binary descriptors, sometimes rivaling even larger floating point representations.

**The GPU architecture and non-blocking programs.** Though the specific architectural designs of GPU processors changes from generation to generation, all have several multiprocessors. A CPU can launch *non-blocking* (parallel) GPU programs on these multiprocessors, referred to as *kernels*. That is, while a kernel is being executed on the GPU, the CPU is free to pursue other tasks and similarly, memory transfers to and from the GPU can take place without blocking either CPU or GPU. This property is extremely important when designing computer vision systems using the GPU: It implies that if the GPU extracts descriptors independently of the CPU, *the CPU is free to perform higher level processing*. Related to the SfM application considered here are optimizations for recovering transformations [11] and/or multiple view stereo for scene structure [9].

Some previous attempts to port descriptors to the GPU used it only for parts of the descriptor extraction process, using the CPU for others and requiring multiple memory transfers between processors [28,36]. This at least partially explains why these attempts showed only modest run time improvements over their original, CPU implementations. As a design goal, we therefore limit the use of the CPU and any communications between it and the GPU when extracting and comparing our descriptors.

**Why LATCH?** LATCH was selected for following reasons.

– **Memory access vs. computation.** The emphasis in GPU processing on raw arithmetic power results in memory access patterns often being the determining factor in performance rather than the actual computation. LATCH requires more processing than pure binary representations (e.g., [8,25,17,1,2]) and therefore requires more CPU time to compute than they do. The memory transfer requirements of LATCH, however, are very similar to these other descriptors and hence it stands to gain more on the GPU.
– **Limited conditional branching.** As mentioned above, GPUs are optimized for processes which have few, if any, conditional branching; under these circumstances, modern GPUs are capable of up to 10 Tera-FLOPS. Most pure binary descriptors are therefore well suited for GPU processing, whereas porting more complex representations to the GPU is less trivial.
– **Binary string comparisons.** LATCH is a binary representation. Like other binary representations, it can be matched using fast Hamming distance comparisons. These can further be performed extremely fast on the GPU.

Finally, as demonstrated in the tests reported by [18], LATCH outperforms other binary descriptors making it ideally suited for our purposes.

## 3.2 Implementing LATCH with CUDA[1]

We have ported the LATCH representation to CUDA 8, building on the original LATCH OpenCV C++ implementation. In all our evaluations, CLATCH representations were extracted from $64 \times 64$ pixel windows, using mini-patches of $8 \times 8$ pixels giving a 64-byte feature vector.

To minimize CPU processing, differently from [18], we use the Features from Accelerated Segment Test (FAST) [24] feature detector. FAST is already available on the GPU as part of the OpenCV [14] library. Given a detected oriented keypoint, $\mathbf{p} = (x, y, \theta)$ we extract LATCH from a $64 \times 64$ intensities window around this point. This process is described next.

**Parallelizing LATCH on the GPU.** GPU kernels include several identical, concurrently-executing, non-interacting *blocks*. Each one consists of groups (*warps*) of 32 threads. In our implementation, a CLATCH kernel sequentially computes 16 descriptors per block. While the region of interest for one interest point is being processed, the next one is prefetched to pipeline the processing.

A single descriptor is extracted by multiple warps in each block. Each warp independently computes sixteen patch triplets, $[\mathbf{P}_{t,A}, \mathbf{P}_{t,1}, \mathbf{P}_{t,2}]$, four at a time, without any explicit synchronization during the main computation. All told, two blocks of 32 warps, each one containing 32 threads (total of 2048 threads) are processed at a time per multiprocessor. This coarse granularity was chosen to maximize performance across a variety of GPU architecture generations.

**Memory optimization.** Given the FAST orientation for an image region, the rotated $64 \times 64$ pixel rectangle is loaded into shared memory as an upright square of single-precision floats. We use texture memory accesses to efficiently load and process these values. Our implementation eliminates bank conflicts, with warp divergences or branches kept to a minimum. Thus, processing proceeds without `if` statements or communications between different warps. This is achieved by strided access patterns of patches and careful padding of shared memory, and is critical to CLATCH's high performance.

Specifically, patch comparisons are performed as follows. A warp simultaneously processes four triplets. Each thread (in a warp of 32 threads) performs two squared-distance comparisons per triplets in the F-norm of Eq. 1. Then, fast warp shuffle operations are used to quickly sum the result from all pixel pairs in a novel, optimal manner. The original LATCH implementation used $7 \times 7$ pixel patches. We use $8 \times 8$ patches instead as this implies 64 values which can be handled concurrently with no extra computation costs. To further optimize this process, instruction level parallelism was exploited by manual loop unrolling and carefully arranging operations to prevent stalls due to data dependency.

**Weighing pixels in LATCH patches.** Each pixel within a patch can optionally be given a unique weight at no overhead. This is due to the GPU's emphasis on cheap fused-multiply-add operations. We use this property to simulate the original LATCH patch size of $7 \times 7$ by setting the relevant weights to zero, obtaining the exact same representation as the original LATCH. Another potential

[1] For brevity, only implementation highlights are provided. For more details, please see the code available from: `www.openu.ac.il/home/hassner/projects/LATCH`.

use for this feature, not tested here, is applying Gaussian weights to patch pixels thereby better emphasizing similarity at the patch center vs. its outer pixels.

## 4 SfM using CLATCH and OpenMVG

LATCH (and consequently CLATCH) were shown to be slightly less accurate than some of the more computationally heavy, floating point descriptors. It is not clear, however, how these differences in accuracy affect the overall accuracy and speed of an entire, descriptor-intensive computer vision system.

To this end, we test CLATCH vs. other descriptors on the challenging task of 3D SfM reconstruction. Our goal is to see how the final reconstruction and the time required to compute it are affected by the choice of descriptor. We use the OpenMVG, multiple view geometry library [20], modifying it to include self-contained CUDA streams and a GPU based, brute force Hamming matcher. These are detailed next.

**CUDA Integration.** The kernel launching mechanism employed by CUDA on its default stream disables concurrent launches of feature detection kernels. We therefore modified the CLATCH descriptor and matching code to exclusively operate off of self-contained streams. Doing so allowed the GPU to concurrently execute feature detection and description kernels across multiple images at once, as well as perform feature matching.

**Descriptor matching on the GPU.** Our tests compare the use of our binary descriptors with existing floating point representations. In all cases, we used the GPU to compute the descriptor distances. Because CLATCH is a binary representation, Hamming distance is used to compute similarity of CLATCH descriptors. To this end, we developed our own GPU based Hamming distance brute force matcher and integrated it into OpenMVG. To provide a fair comparison, distances between floating point representations were computed using the standard OpenCV GPU based L2 distance routine.

Each block of our Hamming-based brute force matching kernel processes half a probe descriptor per thread, though each descriptor is distributed throughout a half warp so that each thread holds parts of 16 probe descriptors. Gallery descriptors are alternatively prefetched into and processed from two shared memory buffers without intermediate synchronization. As the Hamming distance between each pair of probe and gallery descriptors is computed, partial results are distributed through a half warp. This calls for a simultaneous reduction of several independent variables, which minimizes the number of additions and warp shuffles to be performed.

First, each thread halves the number of variables it must reduce by packing two variables into the lower and upper 16 bits of a 32 bit integer. Then, pairs of threads simultaneously exchange their packed variables in a warp shuffle, and sum the result with their original variable. This results in pairs of threads with variables holding identical values. The threads again pair off in the same manner, but exchange and sum a different variable. The same pairs of threads now have two variables with identical values, so the second of each pair of threads overwrites the first packed variable with the second, before each thread discards the second packed variable. This results in every thread in a warp having a unique value in the same variable, which allows efficient participation in subsequent warp shuffles until the reduction is complete. This novel method requires only 16 additions to compare 16 descriptor pairs, while the standard warp reduction pattern would require 80.

## 5 Experiments

**Descriptor extraction run time comparison.** The CLATCH descriptor is identical to LATCH and so their accuracy on different benchmarks are the same. We therefore refer to the original paper for a comparison on standard benchmarks of LATCH and other representations [18].

By using the GPU, CLATCH is much faster to extract. This is demonstrated in Table 1, which provides a comparison of the run times reported for extracting many popular existing feature point descriptors compared to CLATCH. We report also the processor used to extract these representations and a price estimate for the processor in case of GPU based methods.

Evident from the table is that even on affordable GPU hardware, extraction run times are orders of magnitude faster than standard representations on the CPU and even other GPU representations (the only exceptions are the far less accurate CUDA ORB and the floating point representation CUDA SURF). PN-Net [3] in particular, is designed to be a very fast deep learning based descriptor method, yet even with more expensive GPU hardware, it is more than an order of magnitude slower to extract than CLATCH. More importantly, all floating point representations, including CUDA SURF and PN-Net, require more time to match their bigger, real valued representations.

**SfM results.** We use the incremental SfM pipeline implemented in OpenMVG, with its default values unchanged. We compared the following descriptors in our tests: SIFT [19], often the standard in these applications, the deep learning based features, DeepSiam and DeepSiam2Stream from [40], the fast deep feature representations, PN-Net from [3] and our own CLATCH.

All descriptors used the CUDA FAST feature detector with the exception of SIFT which, for technical reasons, used its default DOG based detector. Following incremental SfM, point cloud Densification [5], Mesh Reconstruction [15] and Mesh Refinement [35] were applied to produce the final reconstructions visualized in Fig. 1.

Tests were performed on publicly available sets of high resolution photogrammetry images from [23], which include $5,616 \times 3,744$ (or $3,744 \times 5,616$) pixels in each image. Table 2 summarizes these results, providing the final scene reprojection RMSE and the total time for descriptor extraction, matching and SfM reconstruction. All these tests were run on our GTX 1080 GPU.

Reconstruction run time is dominated by the brute force, nearest neighbor matcher. Hence, the gaps in run times between the different methods are smaller

| Descriptor | Extraction $\mu S$ | GPU |
|---|---|---|
| SIFT [19] | 3290 | - |
| SURF [6] | 2110 | - |
| CUDA SURF [6][1] | 0.9 | GTX 970M ($280 usd) |
| LDA-HASH [30] | 5030 | - |
| LDA-DIF [30] | 4740 | - |
| DBRIEF [34] | 8750 | - |
| BinBoost [16,33] | 3290 | - |
| BRIEF [8] | 234 | - |
| ORB [25] | 486 | - |
| CUDA ORB [25][1] | 0.5 | GTX 970M ($280 usd) |
| BRISK [17] | 59 | - |
| FREAK [1] | 72 | - |
| A-KAZE [2] | 69 | - |
| LATCH [18] | 616 | - |
| DeepSiam [40][2,3] | 6580 | Titan ($650 usd) |
| MatchNet [10][4] | 575 | Titan X ($1,000 usd) |
| CNN3 [26][2] | 760 | Titan Black ($1,100 usd) |
| PN-Net [3][2] | 10 | Titan X ($1,000 usd) |
| Our CLATCH | 0.5 | GTX 970M ($280 usd) |

**Table 1: Run time analysis**. Mean time in microseconds for extracting a single local descriptor. For GPU descriptors we provide also the GPU models used to obtain these results and their estimated price. CPU results were all measured by [18] on their system. [1]CUDA ports for SURF and ORB are implemented in OpenCV [14]; their speeds were measured by us. [2]Run times and hardware specs provided in the original publications. [3]Time for extracting and matching a descriptor pair was reported as $\times 2$ SIFT extraction time. [4]Run time reported in [3].

than those in Table 1. Nevertheless, reconstructions with CLATCH required a fraction of the time for the runner up (PN-Net) and far less than the others.

Reprojection RMSE, is low for all methods and is typically around half a pixel. Although these errors fluctuate between the different methods and scenes, these differences are often below 0.1 pixels. Considering the high resolutions of the input images, these differences are negligible.

Finally, Fig. 1 additionally provides qualitative results, showing rendered views of our reconstructions. Evident from the figure is that despite large differences in run time, qualitatively, the reconstructions appear very similar.

## 6 Conclusions

In descriptor matching intensive application, such as SfM, accuracy per descriptor is sometimes balanced by the speed required to extract and match the descriptors. Taking advantage of this, we present CLATCH, a CUDA port for the LATCH binary descriptor. Although CLATCH descriptor accuracy in standard benchmarks may fall slightly behind other representations, particularly recent deep learning based methods, they are far faster to extract and match. CLATCH thereby provides a fast and accurate alternative means for 3D reconstruction.

From a technical point, an outcome of this work is openly available code for extremely fast feature extraction and matching and a pipeline for SfM allowing convenient interchange of feature descriptors, including deep methods. This

| | Avignon | Bouteville | Burgos | Cognac | Garden St. Jacques | Mirebeau | Murato | Poitiers |
|---|---|---|---|---|---|---|---|---|
| Number of images | 11 | 26 | 9 | 12 | 20 | 22 | 43 | 33 |
| **SfM Scene RMSE** (in pixels) | | | | | | | | |
| SIFT [19] | 0.475 | 0.405 | 0.495 | 0.438 | 0.498 | 0.478 | 0.533 | 0.690 |
| DeepSiam [40] | 0.533 | 0.489 | 0.422 | 0.566 | 0.535 | 0.489 | 0.533 | 0.547 |
| DeepSiam2stream [40] | 0.505 | 0.457 | 0.419 | 0.536 | 0.522 | 0.459 | 0.496 | 0.529 |
| PN-Net [3] | 0.538 | 0.462 | 0.493 | 0.554 | 0.536 | 0.482 | 0.531 | 0.533 |
| Our CLATCH | 0.556 | 0.414 | 0.466 | 0.478 | 0.409 | 0.466 | 0.494 | 0.454 |
| **Total time for descriptor extraction, matching and incremental SfM** (in seconds) | | | | | | | | |
| SIFT [19] | 174.30 | 454.13 | 143.728 | 155.61 | 296.431 | 401.64 | 958.778 | 1206.10 |
| DeepSiam [40] | 172.49 | 596.07 | 130.72 | 146.49 | 347.576 | 416.01 | 943.841 | 812.39 |
| DeepSiam2stream [40] | 269.39 | 922.95 | 226.123 | 301.67 | 628.03 | 739.629 | 1750.535 | 1379.80 |
| PN-Net [3] | 49.56 | 210.60 | 50.12 | 56.02 | 122.29 | 167.51 | 372.28 | 311.18 |
| Our CLATCH | 18.91 | 69.07 | 15.907 | 19.089 | 27.877 | 47.534 | 86.377 | 61.868 |

**Table 2: Quantitative SfM reconstructions.** Results on the eight scenes from [23], comparing various representations with our CLATCH. We report reprojection errors and the time required to extract, match and estimate shape for the various descriptors. All results measured on the same hardware. CLATCH run times are substantially faster than its alternatives despite similar qualitative results (see Fig. 1.)

implementation can be improved in many ways. For one thing, our use of the FAST detector [24] does not provide scale invariance. CLATCH, however, can easily be extracted at multiple scales, potentially improving its accuracy. Use of CLATCH in other applications where descriptors are extracted and matched in large quantities, is also a priority. One particularly appealing example is dense pixel matching [12,31] where the CLATCH may be an alternative to methods such as PatchMatch [5], providing similar run times without compromising spatial smoothness.

## References

1. Alahi, A., Ortiz, R., Vandergheynst, P.: Freak: Fast retina keypoint. In: Proc. Conf. Comput. Vision Pattern Recognition. pp. 510–517. IEEE (2012)
2. Alcantarilla, P.F., Nuevo, J., Bartoli, A.: Fast explicit diffusion for accelerated features in nonlinear scale spaces. In: British Machine Vision Conf. (BMVC) (2013)
3. Balntas, V., Johns, E., Tang, L., Mikolajczyk, K.: PN-Net: Conjoined triple deep network for learning local image descriptors. arXiv preprint arXiv:1601.05030 (2016)
4. Balntas, V., Tang, L., Mikolajczyk, K.: BOLD-binary online learned descriptor for efficient image matching. In: Proc. Conf. Comput. Vision Pattern Recognition. pp. 2367–2375 (2015)
5. Barnes, C., Shechtman, E., Finkelstein, A., Goldman, D.: PatchMatch: a randomized correspondence algorithm for structural image editing. Trans. on Graphics 28(3), 24 (2009)
6. Bay, H., Tuytelaars, T., Van Gool, L.: Surf: Speeded up robust features. In: European Conf. Comput. Vision, pp. 404–417. Springer (2006)
7. Brown, M., Hua, G., Winder, S.: Discriminative learning of local image descriptors. Trans. Pattern Anal. Mach. Intell. 33(1), 43–57 (2011)

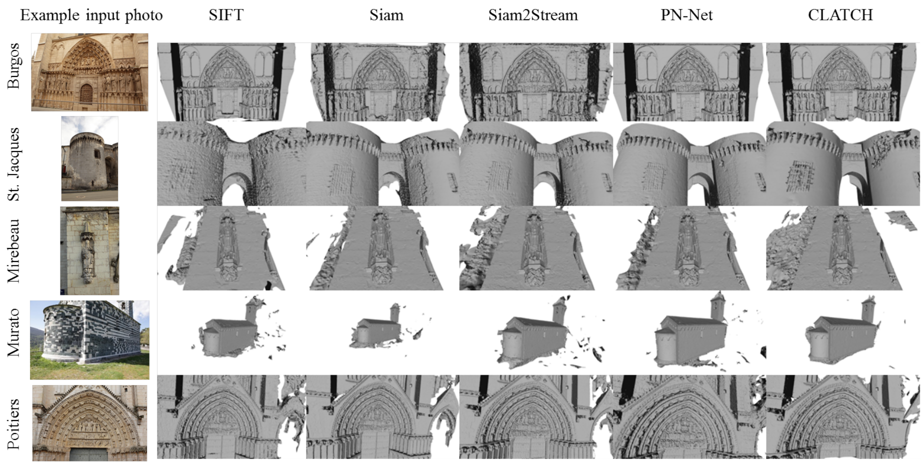

**Fig. 1: Qualitative SfM reconstructions**. Results showing the output of the same SfM pipeline on five of the eights scenes from [23], comparing the use of SIFT [19], Siam and Siam2Stream from [40], PN-Net [3] and our CLATCH. These results show only minor differences in output 3D shapes, despite the substantial difference in run time required for the different representations (see Table 2).

8. Calonder, M., Lepetit, V., Strecha, C., Fua, P.: Brief: Binary robust independent elementary features. In: European Conf. Comput. Vision, pp. 778–792. Springer (2010)
9. Furukawa, Y., Ponce, J.: Accurate, dense, and robust multiview stereopsis. Trans. Pattern Anal. Mach. Intell. 32(8), 1362–1376 (2010)
10. Han, X., Leung, T., Jia, Y., Sukthankar, R., Berg, A.C.: MatchNet: unifying feature and metric learning for patch-based matching. In: Proc. Conf. Comput. Vision Pattern Recognition. pp. 3279–3286 (2015)
11. Hassner, T., Assif, L., Wolf, L.: When standard RANSAC is not enough: cross-media visual matching with hypothesis relevancy. Machine Vision and Applications 25(4), 971–983 (2014)
12. Hassner, T., Liu, C.: Dense Image Correspondences for Computer Vision. Springer (2015)
13. Heymann, S., Müller, K., Smolic, A., Froehlich, B., Wiegand, T.: SIFT implementation and optimization for general-purpose GPU. In: WSCG (2007)
14. Itseez: Open source computer vision library. https://github.com/itseez/opencv (2015)
15. Jancosek, M., Pajdla, T.: Exploiting visibility information in surface reconstruction to preserve weakly supported surfaces. International Scholarly Research Notices 2014 (2014)
16. Lepetit, V., Trzcinski, T., Fua, P., Christoudias, C.M., et al.: Boosting binary keypoint descriptors. In: Proc. Conf. Comput. Vision Pattern Recognition (2013)
17. Leutenegger, S., Chli, M., Siegwart, R.Y.: Brisk: Binary robust invariant scalable keypoints. In: Proc. Int. Conf. Comput. Vision. pp. 2548–2555. IEEE (2011)
18. Levi, G., Hassner, T.: LATCH: learned arrangements of three patch codes. In: Winter Conf. on Applications of Comput. Vision (2016)

19. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vision 60(2), 91–110 (2004)
20. Moulon, P., Monasse, P., Marlet, R., Others: Openmvg. an open multiple view geometry library. https://github.com/openMVG/openMVG
21. Mur-Artal, R., Montiel, J., Tardós, J.D.: ORB-SLAM: a versatile and accurate monocular slam system. Trans. on Robotics 31(5), 1147–1163 (2015)
22. Nowak, E., Jurie, F., Triggs, B.: Sampling strategies for bag-of-features image classification. In: European Conf. Comput. Vision. pp. 490–503. Springer (2006)
23. Perrot, R.: Set of images for doing 3D reconstruction. https://github.com/rperrot/ReconstructionDataSet
24. Rosten, E., Drummond, T.: Machine learning for high-speed corner detection. In: European Conf. Comput. Vision. pp. 430–443. Springer (2006)
25. Rublee, E., Rabaud, V., Konolige, K., Bradski, G.: Orb: an efficient alternative to sift or surf. In: Proc. Int. Conf. Comput. Vision. pp. 2564–2571. IEEE (2011)
26. Simo-Serra, E., Trulls, E., Ferraz, L., Kokkinos, I., Fua, P., Moreno-Noguer, F.: Discriminative learning of deep convolutional feature point descriptors. In: Proc. Int. Conf. Comput. Vision. pp. 118–126 (2015)
27. Simonyan, K., Vedaldi, A., Zisserman, A.: Learning local feature descriptors using convex optimisation. Trans. Pattern Anal. Mach. Intell. (2014)
28. Sinha, S.N., Frahm, J.M., Pollefeys, M., Genc, Y.: GPU-based video feature tracking and matching. In: Workshop on Edge Computing Using New Commodity Architectures. vol. 278, p. 4321 (2006)
29. Strasdat, H., Montiel, J.M., Davison, A.J.: Visual SLAM: why filter? Image and Vision Computing 30(2), 65–77 (2012)
30. Strecha, C., Bronstein, A.M., Bronstein, M.M., Fua, P.: Ldahash: Improved matching with smaller descriptors. Pattern Analysis and Machine Intelligence, IEEE Transactions on 34(1), 66–78 (2012)
31. Tau, M., Hassner, T.: Dense correspondences across scenes and scales. Trans. Pattern Anal. Mach. Intell. (2014), to appear.
32. Terriberry, T.B., French, L.M., Helmsen, J.: GPU accelerating speeded-up robust features. In: Proc. Int. Symp. on 3D Data Processing, Visualization and Transmission (2008)
33. Trzcinski, T., Christoudias, C.M., Lepetit, V.: Learning image descriptors with boosting. Tech. rep., Institute of Electrical and Electronics Engineers (2013)
34. Trzcinski, T., Lepetit, V.: Efficient discriminative projections for compact binary descriptors. In: European Conf. Comput. Vision, pp. 228–242. Springer (2012)
35. Vu, H.H., Labatut, P., Pons, J.P., Keriven, R.: High accuracy and visibility-consistent dense multiview stereo. IEEE transactions on pattern analysis and machine intelligence 34(5), 889–901 (2012)
36. Warn, S., Emeneker, W., Cothren, J., Apon, A.: Accelerating SIFT on parallel architectures. In: Int. Conf. on Cluster Computing and Workshops (Aug 2009)
37. Wu, C.: SiftGPU: A GPU implementation of scale invariant feature transform (SIFT). cs.unc.edu/~ccwu/siftgpu
38. Yang, X., Cheng, K.T.: Ldb: An ultra-fast feature for scalable augmented reality on mobile devices. In: Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on. pp. 49–57. IEEE (2012)
39. Yang, X., Cheng, K.T.: Local difference binary for ultrafast and distinctive feature description. Trans. Pattern Anal. Mach. Intell. 36(1), 188–194 (Jan 2014)
40. Zagoruyko, S., Komodakis, N.: Learning to compare image patches via convolutional neural networks. In: Proc. Conf. Comput. Vision Pattern Recognition (2015)