

# GPU-based Video Feature Tracking And Matching

Sudipta N. Sinha Jan-Michael Frahm Marc Pollefeys

Yakup Genc

Dept. of Computer Science, UNC Chapel Hill

Siemens Corporate Research

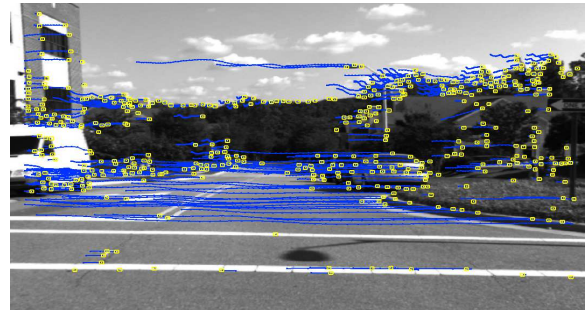
## 1. INTRODUCTION

Extraction and matching of salient feature points in video is important in many computer vision tasks like object detection, recognition and structure from motion. While certain sequential tasks like structure from motion for video [3] require online feature point tracking, others need features to be extracted and matched across frames separated in time (eg. wide-baseline stereo). The increasing programmability and computational power of the graphics processing unit (GPU) present in modern graphics hardware provides great scope for acceleration of computer vision algorithms [5, 1]. We present GPU-based implementations for the popular KLT feature tracker [4] and the SIFT feature extraction [2] algorithm which are 10 to 15 times faster than their optimized CPU counterparts and enable real-time processing of high resolution video.

## 2. KLT TRACKING ON GPU

The KLT tracking algorithm [4] computes displacement of features or interest points between consecutive video frames when image motion is fairly small. Feature selection is done by finding maximas of a saliency measure (minimum eigen-values of the  $2 \times 2$  structure matrix obtained from gradient vectors). It is evaluated over the complete image [4] and a subsequent non-maximal suppression is performed. Assuming a local translational model between subsequent video frames, the feature displacements are computed using Newton's method to minimize the sum of squared distances (SSD) within a tracking window around the feature position in the two images. A multi-resolution KLT tracker allows handling larger image motion while multiple tracking iterations at each scale increases its accuracy. Features tracks are often lost after a few frames of tracking; hence new features are selected in a particular video frame only after tracking features in a few successive frames. This maintains a roughly fixed number of features in the tracker.

**Implementation Details:** GPU-KLT maps these various steps to sets of different fragment programs. The multi-resolution pyramid of the image and its gradients are computed by a series of two-pass separable convolutions performed in fragment programs. The KLT cornerness map is computed in two render passes. The first pass computes the minimum eigen value of the  $2 \times 2$  gradient matrix at each pixel and the two passes together accumulate the cornerness value within a  $7 \times 7$  window centered at each pixel. During feature re-selection, the neighborhood of existing features is invalidated; early Z-culling avoids computations in these image regions. The cornerness map is transferred back to the CPU where non-maximal suppression is done to build the final feature-list. KLT tracking performs a fixed number of tracking iterations at each image resolution starting with the coarsest pyramid level. Each tracking iteration constructs a linear system of equations in two unknowns for each interest point,  $AX = B$  and directly solves them to update



GPU-KLT Timings: 1024 x 768 video, 1000 features.

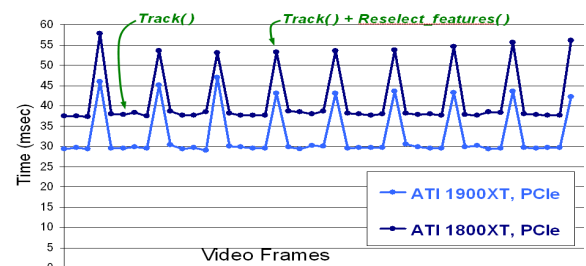


Figure 1: GPU-KLT : real-time tracker for hi-res video.

the estimated displacement. All steps are performed on the GPU. A SSD residual is computed between the two image patches of a particular KLT feature in order to reject features tracked inaccurately. Conditional statements are avoided in fragment programs by tracking a constant number of features and rejecting inaccurate tracks after the final tracking iteration on the GPU and before reading back the feature-list. Our iterative, multi-resolution GPU-KLT tracker runs completely on the GPU contrary to [1] who build the matrices in the system of linear equations on the GPU, perform a readback and solves the equations on the CPU. [1] also do not compare CPU and GPU implementations for accuracy and timings. Our multi-resolution, iterative tracker handles larger image motions and performs accurate tracking on real-time high-resolution video.

**Results:** GPU-KLT was implemented in OpenGL/Cg and tested on the ATI 1800XT and ATI 1900XT graphics cards. Framebuffer objects (FBO) were used for efficiently rendering into texture (16-bit floating point textures were used). Figure 1 shows GPU-KLT feature tracks drawn on an intermediate video frame and compares timings between the CPU and GPU implementations. The ATI 1900XT gave a 15 times speedup over an optimized CPU implementation. The GPU-KLT tracker took 30 msec on an average to track 1000 features in  $1024 \times 768$  resolution video where features were searched within a  $\pm 40$  pixel window. Check results at [http://cs.unc.edu/~ssinha/Research/GPU\\_KLT](http://cs.unc.edu/~ssinha/Research/GPU_KLT)

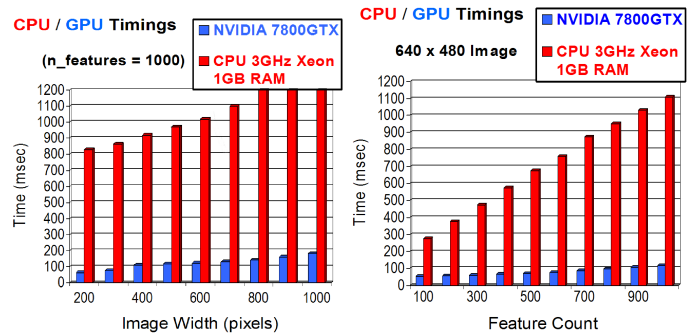
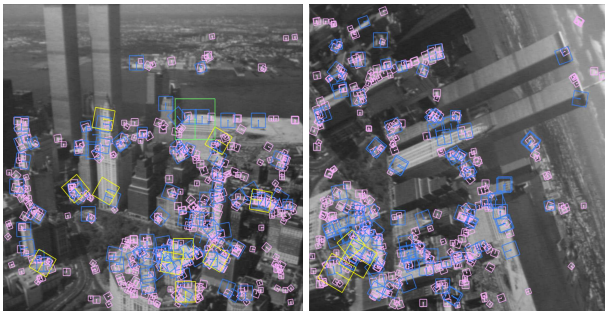


Figure 2: SIFT Feature extraction: CPU vs GPU Timings for a practical range of image-sizes and feature-counts.

### 3. SIFT FEATURE EXTRACTION ON GPU

The Scale Invariant Feature Transform (SIFT) [2] algorithm is a popular candidate for extraction of interest points invariant to translation, rotation, scaling and illumination changes in images. It first constructs a Gaussian scale-space pyramid from the input image while also calculating the gradients and difference-of-gaussian (DOG) images at these scales. Interest points are detected at the local extremas within the DOG scale space. Once multiple keypoints have been detected at different scales, the image gradients in the local region around each feature point are encoded using orientation histograms and represented in the form of a rotationally invariant feature descriptor. The details are described in [2].

**Implementation Details:** The construction of the Gaussian scale space pyramid is accelerated on the GPU using fragment programs for separable convolution. The intensity image, gradients and the DOG values are stored in a RGBA texture and computed in the same pass. Blending operations in graphics hardware are used to find local extremas in the DOG pyramid in parallel at all pixel locations. The Depth test and the Alpha test is used to threshold these keypoints; The local principal curvatures of the image intensity around the keypoint is inspected; this involves computing the ratio of eigenvalues of the  $2 \times 2$  Hessian matrix of the image intensity at that point. The keypoint locations are implicitly computed in image-sized, binary buffers, one for each scale in the pyramid. A fragment program compresses (a factor of 32) the binary bitmap into RGBA data, which is readback to the CPU and decoded there.

At this stage, a list of keypoints and their scales have been retrieved. Since reading back the gradient pyramid (stored in texture memory) to the CPU is expensive, the subsequent steps in SIFT are also performed on the GPU. Gradient vectors near the keypoint location are Gaussian weighted and accumulated inside an orientation histogram by another fragment program. The orientation histogram is read back to the CPU, where its peaks are detected. Computing histograms [1] on the GPU is expensive and doing it on the CPU along with a small readback is a little faster. The final step involves computing 128 element SIFT descriptors. These consist of a set of orientation histograms built from  $16 \times 16$  image patches in invariant local coordinates determined by the associated keypoint scale, location and orientation. SIFT descriptors cannot be efficiently computed completely on the GPU, as histogram bins must be blended to remove quantization noise. Hence we partition this step between the CPU and the GPU. We resample each feature's gradient vector patch, weight them using a Gaussian mask using blending support on the GPU. The resampled and weighted gradient vectors are collected into a tiled texture block which is subsequently transferred back to the CPU and then used to compute the descriptors. This CPU-GPU partition was done to minimize data readback from the GPU since transferring the whole gradient pyra-

mid back to the CPU is impractical. Moreover texture re-sampling and blending are efficient operations on the GPU; hence we perform those steps there. This also produces a compact tiled texture block which can be transferred to the CPU in a single readback.

GPU-SIFT gains a large speed-up in the Gaussian scale-space pyramid construction and keypoint localization steps. The compressed readback of binary images containing feature positions reduces the readback data-size by a factor of 32. The feature orientation and descriptors computation is partitioned between the CPU and GPU in a way that minimizes data transfer from GPU to CPU. Overall a 8-10X speedup is observed compared to CPU versions.

**Results** GPU-SIFT was implemented in OpenGL/Cg and tested on the NVIDIA GeForce 6800GT and 7800GTX cards. Our implementation uses the ping-pong technique between two PBuffer surfaces for efficiently rendering into texture. A texture manager allocates and manages all the data in GPU memory within a single double-buffered PBuffer. The implementation will be upgraded to use FBOs in future. Figure 2 shows GPU-SIFT features extracted from video frames separated in time (camera undergoing rotation) and compares timings between the CPU and GPU implementations for a range of video resolution and feature-count. The NVIDIA 7800GTX gave a 10X speedup over an optimized CPU implementation. GPU-SIFT running on the NVIDIA 7800GTX could extract about 1000 SIFT features from streaming  $640 \times 480$  resolution video at 10 frames/sec on an average. See more results at [http://cs.unc.edu/~ssinha/Research/GPU\\_SIFT/](http://cs.unc.edu/~ssinha/Research/GPU_SIFT/)

### 4. CONCLUSIONS

Both SIFT and KLT have been applied to a wide range of computer vision tasks like stereo matching, object recognition and video mining with quite promising results. We have successfully ported these algorithms to the GPU. Our GPU implementations are considerably faster than optimized CPU versions making it possible to process high resolution video within real-time vision applications.

### 5. REFERENCES

- [1] J. Fung and S. Mann. Openvidia: parallel gpu computer vision. In *MULTIMEDIA*, pages 849–852. ACM Press, 2005.
- [2] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.
- [3] M. Pollefeys, L. V. Gool, M. Vergauwen, F. Verbiest, K. Cornelis, J. Tops, and R. Koch. Visual modeling with a hand-held camera. *IJCV*, 59(3):207–232, 2004.
- [4] C. Tomasi and T. Kanade. Detection and tracking of point features. Technical Report CMU-CS-91-132, CMU, 1991.
- [5] R. Yang and M. Pollefeys. Multi-resolution real-time stereo on commodity graphics hardware. In *CVPR'03*, pages 211–217.