Fast Gain-Adaptive KLT Tracking on the GPU

Christopher Zach, David Gallup, Jan-Michael Frahm
University of North Carolina
Chapel Hill, NC
{cmzach,gallup,jmf}@cs.unc.edu

Abstract

High-performance feature tracking from video input is a valuable tool in many computer vision techniques and mixed reality applications. This work presents a refined and substantially accelerated approach to KLT feature tracking performed on the GPU. Additionally, a global gain ratio between successive frames is estimated to compensate for changes in the camera exposure. The proposed approach achieves more than 200 frames per second on state-of-the art consumer GPUs for PAL (720×576*) resolution data, and delivers real-time performance even on low-end mobile graphics processors.*

1. Introduction

Tracking of simple corner-like features in a video sequence [10, 15] is still often applied due to its simplicity and its excellent runtime performance. In particular, augmented reality applications (e.g. using the OpenVIDIA framework [6, 4]) and high performance scene reconstruction from streamed video [11] require very fast detection of interest points and the subsequent search for potential correspondences from a video source.

Using the GPU accelerates many image processing and computer vision methods due to their highly data-parallel nature (e.g. [20, 2, 18, 5, 21]). In addition to the parallelism provided by a set of fragment processors working synchronously on different inputs, the fast texture mapping units, which enable filtered access to 2D and 3D image data, can be exploited as well. Especially, multi-view reconstruction methods further benefit from this GPU-specific feature [19, 3, 9].

One well-known implementation of a GPU-based KLT feature tracker is described in Sinha et al. [16, 17]. The reported runtimes for GPU-KLT tracking applied on 800×600 pixel images and maintaining up to 1000 tracked features are about 26msec using ATI graphics hardware and 40msec on NVidida GPUs. Although these figures are impressive and indicate realtime performance at typical video resolutions, it turns out that this implementation does not scale well on the newest generation hardware, e.g. on the Geforce 8 series. This means that the significantly increased computing power of the newer GPUs is only partially reflected in higher frames rates. One reason for the unexpected poor scalability is the heavy use of point-rendering in several stages of the implementation. Note, that Sinha's GPU-KLT approach was made available to the public as open source software.[1]

In this paper we propose a GPU based KLT tracker with a significantly improved performance and simultaneous estimation of the gain of the camera as shown in [8]. In order to compensate for gain differences, we use a simple linear camera response model to estimate the gain ratio. The library source is made publicly available[2]; it is functionally equivalent replacement for Sinha's GPU-KLT, but not an API equivalent implementation.

2. Data-Parallel Tracking and Simultaneous Gain Estimation

In this section we sketch the gain-adaptive tracking approach proposed in [8] at first and subsequently describe the necessary modifications required to obtain a data-parallel procedure suitable for a GPU implementation.

2.1. Review of Gain Adaptive Realtime Stereo Streaming

In order to straighten the description of our approach and to highlight the main differences to earlier work, we outline the combined tracking and gain estimation procedure proposed in [8]. The simplifying assumption is, that a multiplicative model for camera gain variations over time is sufficient, i.e. the pixel intensities of adjacent frames are related by

$$I^{(t+1)}(\mathbf{x}_i^{(t+1)}) = \beta I^{(t)}(\mathbf{x}_i^{(t)}), \tag{1}$$

[1] http://cs.unc.edu/~ssinha/Research/GPU_KLT
[2] http://cs.unc.edu/~cmzach/opensource.html

where $I^{(t)}$ and $I^{(t+1)}$ are the intensity images at successive timestep t and $t+1$, and $\mathbf{x}_i^{(t)}$ and $\mathbf{x}_i^{(t+1)}$ denotes a corresponding pair of pixels. The gain ratio β is applied uniformly for the whole image $I^{(t+1)}$. In the following we will abbreviate the notations and use I and J for $I^{(t)}$ and $I^{(t+1)}$, respectively. Further, $\mathbf{x}_i^{(t)}$ is shortened to $\mathbf{x}_i$, and $\mathbf{x}_i^{(t+1)}$ to $\mathbf{y}_i$.

Then the application of the first order Taylor expansion on J, $J(\mathbf{y}+\mathbf{d}) \approx J(\mathbf{y}) + \langle \nabla J, \mathbf{d} \rangle$, yields to the modified brightness constancy assumption,

$$\beta I(\mathbf{x}) - J(\mathbf{y}) - \langle \nabla J, \mathbf{d} \rangle = 0. \tag{2}$$

This brightness constancy equation is enforced in a least-squares sense for a neighborhood $\mathcal{W}_i$ of a tracked feature point $\mathbf{x}_i$, hence the the position update $\mathbf{d}_i$ and the unknown, but uniform gain parameter β are the minimizers of

$$E(\mathbf{d}_i, \beta) = \sum_i \sum_{\mathcal{W}_i} \left(\beta I(\mathbf{x}) - J(\mathbf{y}) - \langle \nabla J, \mathbf{d} \rangle\right)^2. \tag{3}$$

Deriving this expression with respect to the unknown yields to the following linear equation system:

$$\begin{bmatrix} \mathbf{U} & \mathbf{w} \\ \mathbf{w^T} & \lambda \end{bmatrix} \begin{bmatrix} \mathbf{d} \\ \beta \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ c \end{bmatrix}, \tag{4}$$

where $\mathbf{d} = [\mathbf{d}_1, \ldots, \mathbf{d}_n]^T$ contains all position updates, $\mathbf{U}$ is a diagonal block matrix with the respective structure tensor along the diagonal, $\mathbf{w}$ and $\mathbf{b}$ are suitable vectors, and λ is a suitable scalar. After one step of Gaussian elimination (also referred as Schur complement), β can be directly computed,

$$(\mathbf{w^T U^{-1} w} + \lambda)\beta = \mathbf{w^T U^{-1} b} + c. \tag{5}$$

Note, that the scalar β depends on all structure tensors $(\nabla J)^T \nabla J$. On a data-parallel computing device, determining the gain ratio β requires an expensive reduction operation. With the knowledge of β, the position updates are determined by the traditional KLT equations applied on βI and J.

2.2. Data Parallel Extension

The symmetric version of the brightness constancy constraint [1],

$$\begin{aligned} 0 &= \beta I\big(\mathbf{x} - \frac{\mathbf{d}}{2}\big) - J\big(\mathbf{y} + \frac{\mathbf{d}}{2}\big) \\ &= (\beta^0 + d\beta) I\big(\mathbf{x} - \frac{\mathbf{d}}{2}\big) - J\big(\mathbf{y} + \frac{\mathbf{d}}{2}\big) \\ &\approx (\beta^0 + d\beta) I(\mathbf{x}) - J(\mathbf{y}) - \left\langle \frac{\beta^0 \nabla I + \nabla J}{2}, \mathbf{d} \right\rangle, \end{aligned}$$

is generally reported to be superior to the one-sided formulation Eq 2, hence we utilize this computationally more expensive approach. Note, that we perform the Taylor expansion of $\beta I(\mathbf{x} - \frac{\mathbf{d}}{2})$ at some previous estimate β^0. Knowledge of both image gradients, $\nabla I(\mathbf{x})$ and $\nabla J(\mathbf{y})$ enables a better estimate for β, since under noise-free conditions we have

$$\beta |\nabla I| = (\beta^0 + d\beta)|\nabla I| = |\nabla J| \tag{6}$$

due to the assumption of a linear radiometric response function. Consequently, we can combine the brightness constancy requirement and the gradient equality into the following extended least squares energy to minimize:

$$\begin{aligned} E(\mathbf{d}_i, d\beta) = \gamma \sum_i \sum_{\mathcal{W}_i} \left((\beta^0 + d\beta)|\nabla I| - |\nabla J|\right)^2 \\ + \sum_i \sum_{\mathcal{W}_i} \left((\beta^0 + d\beta) I(\mathbf{x}_i) - J(\mathbf{y}_i) - \langle \nabla \bar{I}, \mathbf{d}_i \rangle\right)^2, \end{aligned} \tag{7}$$

where we use $\nabla \bar{I}$ as abbreviation for $(\beta^0 \nabla I + \nabla J)/2$. γ is a user-specified weighting parameter determining the influence of the gradient term. Still, the occurrence of the global gain ratio β^0 and its incremental update, $d\beta$, prohibits the fully accelerated utilization of data-parallel computing devices. Obviously, using one global unknown β is equivalent to utilizing local counterparts β_i known to each feature patch and adding suitable equality constraints, $\beta_i = \beta_j \forall i, j$. Incorporating the exact Lagrange multipliers still requires an undesired global reduction step, hence methods using user supplied Lagrange multipliers (like the penalty methods or augmented Lagrangian approaches) are more practical. For simplicity, we just adopt the quadratic penalty method using $\beta_i = \beta_j$ for $j \in \mathcal{N}(i)$ as constraints, where $\mathcal{N}(i)$ denotes a set of (not neccessarily spatial) neighboring indices for i. Replacing β_i by its incremental update $\beta_i^0 + d\beta_i$, and using the quadratic penalty method with parameter μ, the following expression is minimized:

$$\begin{aligned} E(\mathbf{d}_i, d\beta_i) = \gamma \sum_i \sum_{\mathcal{W}_i} \left((\beta_i^0 + d\beta)|\nabla I| - |\nabla J|\right)^2 \\ + \sum_i \sum_{\mathcal{W}_i} \left((\beta_i^0 + d\beta_i) I(\mathbf{x}_i) - J(\mathbf{y}_i) - \langle \nabla \bar{I}, \mathbf{d}_i \rangle\right)^2 \\ + \mu \sum_i \sum_{j \in \mathcal{N}(i)} \left(\beta_i^0 + d\beta_i - (\beta_j^0 + d\beta_j)\right)^2. \end{aligned} \tag{8}$$

Taking the derivatives with respect to $\mathbf{d}_i$ and $d\beta_i$, a system of linear equations with a symmetric and positive definite system matrix A is obtained. The matrix A has a specific shape: along its diagonal it contains 3×3 blocks of symmetric and positive definite matrices. We will denote this block diagonal matrix by P. The remaining entries are gathered in the matrix N, such that $A = P - N$. Since P is easy to invert, a *block Jacobi method* is an appropriate choice for a data-parallel approach to solve this linear system, i.e. the next estimate for the vector of unknowns,

$\mathbf{z} = [\mathbf{d}_1, d\beta_1, \ldots, \mathbf{d}_N, d\beta_N]$,

$$\mathbf{z}^{(k+1)} = P^{-1}\left(\mathbf{b} + N\mathbf{z}^{(k)}\right), \tag{9}$$

where $\mathbf{b}$ denotes the right hand side of this system. In order to have (global) convergence, the update needs to be a contraction, i.e. $\|P^{-1}(\mathbf{b} + N\mathbf{z}_1) - P^{-1}(\mathbf{b} + N\mathbf{z}_2)\| = \|P^{-1}N\mathbf{z}_1 - P^{-1}N\mathbf{z}_2\| < \|\mathbf{z}_1 - \mathbf{z}_2\|$ for all $\mathbf{z}_1, \mathbf{z}_2$. We show in the appendix, that the update in Eq. 9 is in fact a contraction. Note, that the resulting numerical scheme closely resembles a homogeneous diffusion procedure for β_i, and the quadratic penalty term acts like a regularization force.

3. Implementation

In this section we provide details on our current implementation. Although it is currently very common to use NVidias *Compute Unified Device Architecture (CUDA)* for non-graphical computations on the GPU, our own experience indicates, that shader-based implementations are still superior in many cases. In particular, initial performance measurements of CUDA and Cg/OpenGL version showed, that the latter one is substantially faster than its CUDA counterpart. Note, that generally KLT tracking of individual features is highly parallel and does not require communication between threads processing individual tracks. Hence, the main feature of CUDA – data transfer between threads using fast shared memory – is not exploited. Additionally, the use of Cg/OpenGL allows older graphics hardware to run our GPU KLT implementation, which slightly increases its value.

Our implementation consists of three main modules, namely image pyramid generation, corner detection and the feature position update.

3.1. Generation of Image Pyramids

The creation of the respective image pyramid for the grayscale input frames uses essentially a straightforward implementation. As a preprocessing step the current frame is smoothed with a small odd binomial kernel before building the full pyramid. The first horizontal pass convolves the source image with the binomial kernel and its derivative, yielding the smoothed image and its horizontal derivative as two 32-bit floats packed in four 16-bit channels. The subsequent vertical pass generates three 16-bit float channels, the smoothed intensity image and its x- and y-derivatives. The remaining levels of the image pyramid are obtained by recursive filtering with an even binomial kernel.

In terms of memory bandwidth it appears to be more efficient not to generate the image derivatives while building the pyramid and to compute the derivatives on demand in the tracking step. In practice we observed (for Cg and CUDA implementations), that such an approach is substantially slower than the above one. We conjecture that the increased size of local memory required in the latter approach (instead of a few registers) is the cause for this runtime behaviour.

Image pyramids using 16-bit floats store the smoothed original image without loss in accuracy (since we employ a small binomial kernel). The reduction in precision in coarser pyramid levels is not a major concern, but increases the overall performance by approximately 30%.

3.2. Corner Detection and Redetection

Good pixels to track have a rich structure tensor [15], i.e. the smaller eigenvalue of

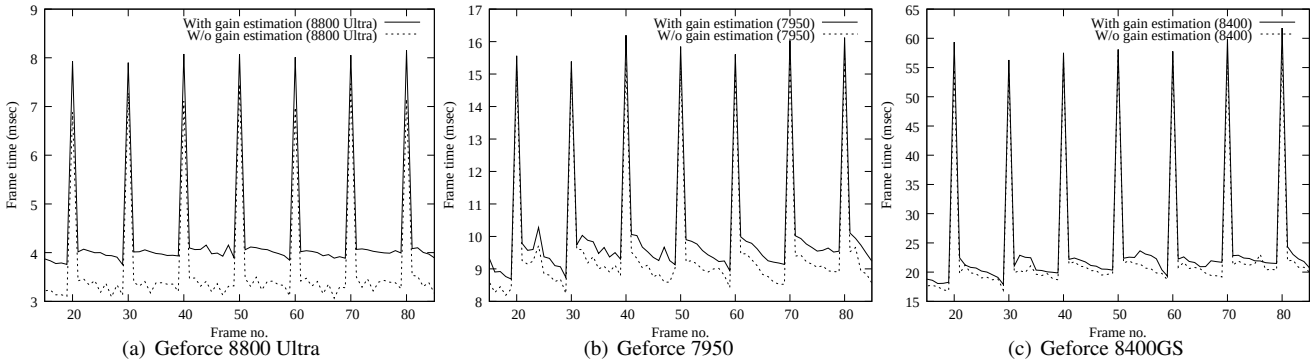
$$\sum_{\mathbf{x}\in\mathcal{W}} (\nabla I(\mathbf{x}))^T \nabla I(\mathbf{x}) \tag{10}$$

is sufficiently large. If both eigenvalues are close to zero, the respective window region is very homogeneous. If only one eigenvalue is substantially larger than zero, then we are facing an edge and only the normal flow can be calculated by a purely local approach. In order to track single point features, reliable corners need to be extracted. The generation of the image pyramid already provides us with the (smoothed) image gradients required for the structure tensor computation, and suitable box filtering exploiting the separability yields the complete structure tensor. The smaller eigenvalue, i.e. the cornerness, of a 2-by-2 matrix can be easily computed by a fragment shader, which additionally performs a thresholding operation and discards potential corners very close to the image border. The result of this operation is a floating point texture (encoded in an 8-bit per channel RGBA image using a packed representation[3]), which has positive values at potential corner pixels and zeros otherwise.

In order to avoid concentration of extracted features at few highly textured image regions, a non-maximum suppression procedure is subsequently applied. We employ a modified column-wise and row-wise max-convolution procedure: after each of the two passes, a positive value in the resulting image buffer indicate that the respective position is a local maximum, whereas negative values depict pixels, where the maximum was propagated from the corresponding neighborhood. Thus, each pass of the convolution assigns the value largest in magnitude within the support window to the current position.

Keeping already detected and successfully tracked features during a redetection step is very simple: rendering points for the still valid features using a negative "color" with large magnitude precedes the non-maximum suppression step, hence those features are retained. It turns out that

[3]Only available on hardware supporting the `NV_FRAGMENT_PROGRAM_OPTION` OpenGL extenstion, which includes instructions to treat 4 8-bit values as one floating point value and vice versa.



(a) Geforce 8800 Ultra (b) Geforce 7950 (c) Geforce 8400GS

Figure 1. Runtimes per frame with and without gain estimation on different GPUs for PAL resolution (720×576).

rendering of such small point sets does not affect the overall performance.

At this stage the GPU has efficiently determined a binary mask for the full image indicating additional corner points suitable for further tracking. Generating a compact list representation of the respective feature positions used in the subsequent tracking steps is known to be a non-trivial task for data-parallel devices and usually referred as *stream compaction* [7]. Note, that downloading the binary mask to the host (CPU) memory and to perform the (then trivial) stream compaction operation by the CPU is expensive, since only vew few pixels in the mask indicate a corner positions (e.g. typically less than 1000 corners in a 1024x768 image). Hence, we employ a GPU-based stream compaction method, which is recently an active research topic [22, 13, 14].

Currently we use the histogram pyramid approach proposed in [22], since it is designed to operate in a shader-based environment like Cg/OpenGL. Additionally, the authors claim, that their approach is significantly faster than geometry shaders applied for stream compaction and even faster than CUDA versions. Essentially, stream compaction using histogram pyramids first computes a full mipmap pyramid by successive parallel summation of 2-by-2 pixels similar to sum reduction by recursive doubling. Thus, the total number of relevant pixels, i.e. newly detected feature points, can be reported immediately from the coarsest level of the pyramid (which consists of a single pixel). In a second phase the compact list of positions is generated by a hierarchical search using the previously generated histogram pyramid. Finally, the minimal number of feature positions is transferred to the host (main) memory for further processing.

The layout of feature tracks in video (texture) memory is two-dimensioal, e.g. a 32×32 texture image if 1024 features are maintained at most. In order to avoid a spatial bias in gain estimation (see below), the vector of newly detected features is randomly permuted.

3.3. Update of Feature Positions

Incorporation of gain ratio estimation replaces the typical 2×2 linear system on the displacement updates by a 3×3 one. Hence, the corresponding fragment shader mainly gathers the pixels and derivatives in the floating windows and constructs the respective linear equation system and its right hand side. The incremental update of the displacement and the gain ratio is computed using Cramer's rule. The block Jacobi updates uses the 8 neighboring values of β_j, which are not spatially related in the image due to earlier random shuffling.

The output of the fragment shader is a 3-vector of floats consisting of the refined position $\mathbf{d}_i$ and the updated gain ratio β_i. Note, that there are several conditions resulting in invalidating features: the updated position may be outside the image region, the image residual is too large or the KLT iterations did not converge. In all these cases the feature track is signaled as invalid by using particular values.

In order to avoid weak local minima, the position and gain ratio updates are embedded into a coarse-to-fine scheme using the image pyramids as described in Section 3.1. As in Sinha's GPU-KLT implementation we use the 3rd and 0th (base) level of the pyramid and skip intermediate ones.

The quadratic penalty parameter μ enforcing consistency among all values of β_i is initially set to a user-supplied value μ_0 and increased in every iteration by a factor τ, i.e. $\mu^{(k+1)} = \tau\mu^{(k)}$. Thus, the magnitude of μ grows exponentially with the iterations unless $\tau = 1$.

In addition to the gain-adaptive KLT version we implemented the traditional KLT approach assuming constant exposure settings and requiring only the solution of a 2×2 system for every tracked feature and iteration.

4. Experimental Results

We have three platforms available for extensive timing experiments. One platform is a desktop PC with an NVidia Geforce 8800 Ultra, another PC is equipped with a dual

(a) (b) (c) (d)

Figure 2. Several frames from a video captured with auto-exposure enabled.

core Geforce 7950 (although only one core of the GPU is utilized), and finally we performed timing measurements on a laptop with a builtin NVidia Geforce 8400 GS. Figures 1(a)–(c) show the observed frame times on the respective hardware for 720×576 (PAL) video input (which has constant exposure to allow a fair timing comparison of traditional and gain-adaptive tracking). In order to measure the pure tracking time, the video frames are prefetched and cached in main memory. The 3rd and the base level of the image pyramids are used, and 5 iterations of the position/gain update are performed on each level. $\gamma = 1$ is hold constant in all experiments. Due to rejection of some features the tracking time between redetection steps (performed every 10 frames) marginally descreases. In summary, KLT tracking with gain estimation runs at 216 frames per second on average, and tracking without gain adaption is slightly faster at 260 fps (on the Geforce 8800 Ultra). The Geforce 7950 and 8400GS GPUs achieve about 100 and 40 frames per second, respectively.

The "spikes" in the runtime graphs in Figure 1 appearing every 10 frames correspond to the combined tracking and redetection steps. Apparently, different hardware has different performance penalties for feature redetection: it is approximately 60% of the tracking time on the Geforce 7950 (Figure 1(b), increasing the frame time from about 10msec to 16ms), but almost 200% on a NVidia Geforce 8400 GS (Figure 1(c)).

We tested the capability of simultaneous gain estimation on another video with camera auto-exposure enabled (Figure 2). The camera reported gain ratios and the estimated ones (taken as the mean of the individual estimated values) are plotted in Figure 3. We reckon that our recovered gain ratio fit the ground truth better than the one presented in [8]. Figure 4 displays the gain ratios for an interesting (i.e. highly varying) section of the video obtained at different settings for the quadratic penalty parameter μ: in particular, the mean gain ratios are depicted for $\mu = 0$ (independent local gain estimation), $\mu = 200$ (constant penalty parameter for all iterations) and exponentially growing $\mu^{(k)} = \tau^k \mu_0$, with $\mu_0 = 40$ and $\tau = 2$. Using only locally estimated gain ratios ($\mu = 0$) is clearly inferior to the regularized versions. Finally, the standard deviation of the gain ratios reported for each patch are depicted in Figure 5. Since we do not iterate the block Jacobi method until convergence, there is still substantial variance observable in the reported β_i. Note that the deviation consistently increases for frames 100 to 150, since fewer features are detected in this section due to fast camera movement.

5. Future Work

Tracking features from video sequences provides a set of putative matches with usually high inlier rates, but further processing – like correspondence verification and pose estimation – are still mandatory. Incorporating our tracking implementation into a more complete structure from motion and/or mixed reality pipeline is open for future work. In particular, a comparable fast feature track verification approach is worthwhile for future investigations. Another area of research and application is the utilization of high-performance tracking in multiple camera setups, where several video streams need to be processed simulateneously.

Appendix: Convergence of the Block Jacobi Method

In the appendix we briefly discuss the convergence of the block Jacobi method applied on the linear system obtained by computing the stationary points of Eq. 8. Of interest here is only the system matrix A consisting of the coefficients for the unknowns $\mathbf{d}_i$ and $d\beta_i$. We order the unknowns, such that columns $3i$ and $3i+1$ correspond to $\mathbf{d}_i$, and column $3i+2$ represents $d\beta_i$, for $i = 1, \ldots, K$. In matrix U we gather all coefficients induced by deriving Eq. 7 with respect to the unknowns. This matrix is symmetric, positive definite and block diagonal with 3×3 blocks along the main diagonal (under the assumption of proper corner features with non-degenerate structure tensors).

Further, the matrix D is defined as $D_{3i,3i} = \mu|\mathcal{N}(i)|$ and zero otherwise. In our implementation, $|\mathcal{N}(i)|$ is 8. Finally, the matrix N is given by $N_{3i,3j} = \mu$ iff $j \in \mathcal{N}(i)$. All other elements of N are zero. Thus, we can write the full system matrix as $A = U + D - N$.

In terms of the update equation, Eq. 9, we have $P = U + D$, and N in Eq. 9 coincides with our definition here. The convergence of block Jacobi methods can be shown using

the following theorem (e.g. [12], Property 4.1): Let $A = P - N$ with A and P symmetric and positive definite. If $2P - A = P + N$ is positive definite, then the iterative method Eq. 9 is convergent for any initial value $\mathbf{z}^{(0)}$.

It remains to show that $P + N = U + D + N$ is positive definite. Recall that U is positive definite, and $D + N$ is diagonally dominant with positive entries in the diagonal. Hence, by the Gershgorin circle theorem, $D + N$ is at least positive semidefinite. In total, we have $P + N$ as the sum of a positive definite and a positive semidefinite matrix, thus $P + N$ is positive definite.

References

[1] S. Birchfield. Derivation of Kanade-Lucas-Tomasi tracking equation. Unpublished, 1997.

[2] P. Colantoni, N. Boukala, and J. D. Rugna. Fast and accurate color image processing using 3D graphics cards. In *Proc. of Vision, Modeling and Visualization 2002*, 2003.

[3] N. Cornelis and L. Van Gool. Real-time connectivity constrained depth map computation using programmable graphics hardware. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1099–1104, 2005.

[4] J. Fung, S. Mann, and C. Aimone. OpenVIDIA: Parallel GPU computer vision. In *Proceedings of the ACM Multimedia 2005*, pages 849–852, 2005.

[5] S. Heymann, K. Müller, A. Smolic, B. Fröhlich, and T. Wiegand. Sift implementation and optimization for general-purpose gpu. In *Proceedings of WSCG*, pages 317–322, 2007.

[6] R. Hill, J. Fung, and S. Mann. Reality window manager: A user interface for mediated reality. In *Proceedings of the 2004 IEEE International Conference on Image Processing (ICIP2004)*, pages 24–27, 2004.

[7] D. Horn. *GPU Gems 2*, chapter Stream Reduction Operations for GPGPU Applications, pages 621–636. Addison Wesley, 2005.

[8] S. J. Kim, D. Gallup, J.-M. Frahm, A. Akbarzadeh, Q. Yang, R. Yang, D. Nister, and M. Pollefeys. Gain adaptive real-time stereo streaming. In *Proc. Int. Conf. on Computer Vision Systems (ICVS)*, 2007.

[9] P. Labatut, R. Keriven, and J.-P. Pons. A GPU implementation of level set multiview stereo. In *International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT)*, 2006.

[10] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *International Joint Conference on Artificial Intelligence (IJCAI81)*, pages 674–679, 1981.

[11] M. Pollefeys, D. Nister, J.-M. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S.-J. Kim, P. Merrell, C. Salmi, S. Sinha, B. Talton, L. Wang, Q. Yang, H. Stewenius, R. Yang, G. Welch, and H. Towles. Detailed real-time urban 3d reconstruction from video. *Int. Journal of Computer Vision*, special issue on Modeling Large-Scale 3D Scenes, 2008.

[12] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical Mathematics*. Springer, 2000.

[13] D. Roger, U. Assarsson, and N. Holzschuch. Efficient stream reduction on the GPU. In *Workshop on General Purpose Processing on Graphics Processing Units*, 2007.

[14] S. Sengupta, M. Harris, Y. Zhang, and J. D. Owens. Scan primitives for GPU computing. In *Graphics Hardware 2007*, pages 97–106, 2007.

[15] J. Shi and C. Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, Seattle, jun 1994.

[16] S. N. Sinha, J.-M. Frahm, M. Pollefeys, and Y. Genc. GPU-based video feature tracking and matching. In *Workshop on Edge Computing Using New Commodity Architectures (EDGE 2006)*, 2006.

[17] S. N. Sinha, J.-M. Frahm, M. Pollefeys, and Y. Genc. Feature tracking and matching in video using programmable graphics hardware. *Machine Vision and Applications (MVA)*, 2007.

[18] K. Sugita, T. Naemura, and H. Harashima. Performance evaluation of programmable graphics hardware for image filtering and stereo matching. In *Proceedings of ACM Symposium on Virtual Reality Software and Technology 2003*, 2003.

[19] R. Yang and M. Pollefeys. Multi-resolution real-time stereo on commodity graphics hardware. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 211–217, 2003.

[20] R. Yang and G. Welch. Fast image segmentation and smoothing using commodity graphics hardware. *Journal of Graphics Tools*, 7(4):91–100, 2002.

[21] C. Zach, T. Pock, and H. Bischof. A duality based approach for realtime TV-L^1 optical flow. In *Annual Symposium of the German Association for Pattern Recognition (DAGM)*, 2007.

[22] G. Ziegler, A. Tevs, C. Theobalt, and H.-P. Seidel. On-the-fly point clouds through histogram pyramids. In *11th International Fall Workshop on Vision, Modeling and Visualization 2006 (VMV2006)*, pages 137–144, 2006.

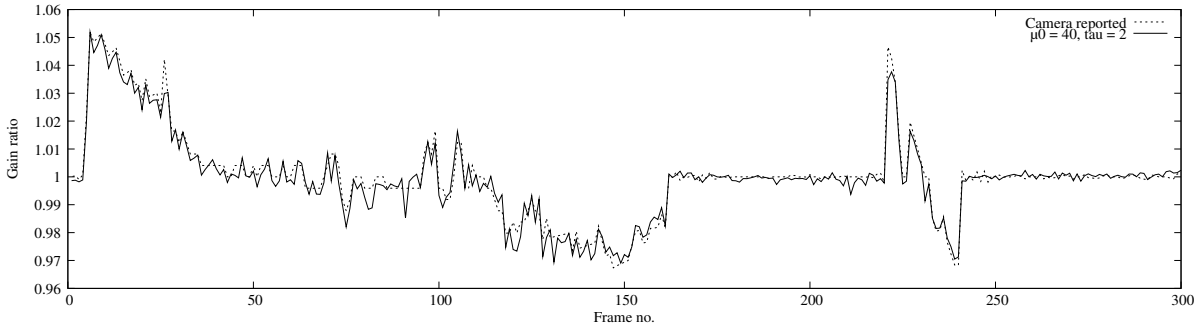


Figure 3. Comparison of camera reported gain ratios (dashed line) with the estimated ones using the indicated parameters (solid line, $\mu_0 = 40, \tau = 2$).

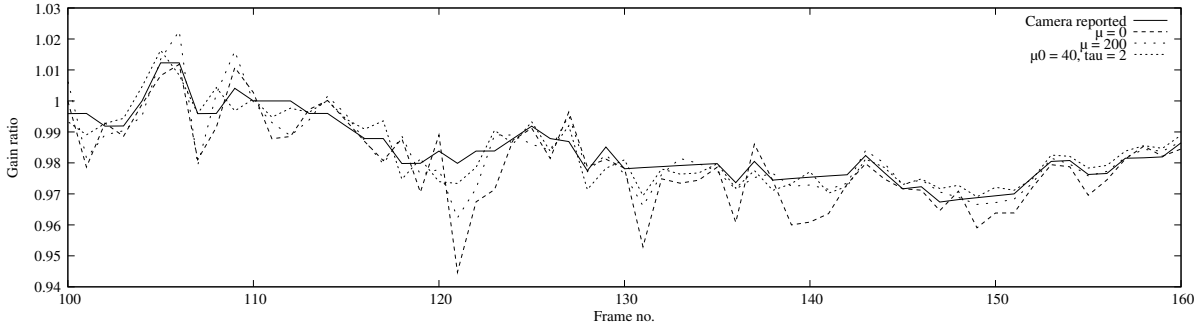


Figure 4. Detail view on the camera reported gain ratios and estimated ones using different parameter settings.

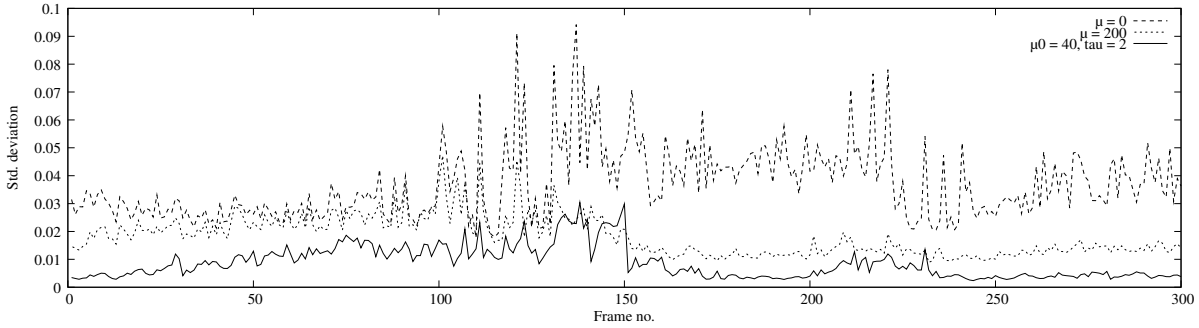


Figure 5. Standard deviations of the estimated gain ratios at different settings for μ.