GPU TV-L^1 Optical Flow

1 Problem statement

Determining optical flow, the pattern of apparent motion of objects caused by the relative motion between observer and objects in the scene, is a fundamental problem in computer vision. Given two images, goal is to compute the 2D motion field - a projection of 3D velocities of surface points onto the imaging surface. Optical flow can be used in a wide range of higher level computer vision tasks, from object tracking and robot navigation to motion estimation and image stabilization.

There is a real need for shortening the required computational time of optical flow for use in practical applications such as robotics motion analysis and security systems. With the advances in utilizing GPUs for general computation, its become feasible to use more accurate (but computationally expensive) optical flow algorithms for these practical applications.

With that in mind, we propose to implement an improved L1-norm based total-variation optical flow (TV-L1) on the GPU.

2 Related work

Graphical Processing Units (GPUs) were originally developed for fast rendering in computer graphics, but recently high level languages such as CUDA [1] have enabled general-purpose GPU computation (GPGPU). This is useful In the field of computer vision and pattern recognition, and [5] demonstrate this by implementing the Horn-Schunck method on GPUs using CUDA.

Two well known variational approaches to computing the optical flow are the point based method presented in Horn and Schunck [3], and the local patch based method of Lucas and Kanade [4]. Our approach is an improvement of [3] that replaces the squared measure of error with an L^1 norm. The highly data-parallel nature of the solution lends itself well to a GPU implementation.

3 Approach

The standard Horn-Schunck algorithm minimizes the following energy equation

$$\min_u \left\{ \begin{array}{l} \int_\Omega |\nabla u_1|^2 + |\nabla u_1|^2 d\Omega + \\ \lambda \int_\Omega (I_1(x + u(x)) - I_0(x))^2 d\Omega \end{array} \right\}$$

where the first integral is known as the regularization term, and the second as the data term. An iterative solution to the above optimization problem is given in [3] as

$$u_x = \bar{u}_x - I_x \frac{[I_x \bar{u}_x + I_y \bar{u}_y + I_t]}{\alpha^2 + I_x^2 + I_y^2}$$

$$u_y = \bar{u}_y - I_y \frac{I_x \bar{u}_x + I_y \bar{u}_y + I_t}{\alpha^2 + I_x^2 + I_y^2}$$

where $\mathbf{u} = (u_x, u_y)$ is the flow field, $\bar{\mathbf{u}}$ is a weighted local average of the flow, I_x, I_y, and I_t are the image gradients in x, y and t respectively, and $alpha$ is a small constant close to zero. These two equations come from solving the Lagrangian optimization equations associated with the energy function, and using a finite differences approximation of the Laplacian of the flow.

The TV-L^1 approach optimizes the following equation

$$\min_u \left\{ \int_\Omega \lambda |I_0(x) - I_1(x + u(x))| + |\nabla u| \right\} d\Omega$$

which replaces the squared error term with an L^1 norm. Wedel et al. [6] propose an iterative solution that alternatively updates the flow and an auxiliary variable $\mathbf{v}$. For a fixed $\mathbf{v}$, $\mathbf{u}$ is updated as

$$\mathbf{u} = \mathbf{v} + \alpha \, \textbf{div} \, \mathbf{p}$$

where $\mathbf{p}$ is iteratively defined as

$$\tilde{\mathbf{p}}^{n+1} = \mathbf{p}^n + \frac{\tau}{\alpha} \nabla \mathbf{u}$$

$$\mathbf{p}^{n+1} = \frac{\tilde{\mathbf{p}}^{n+1}}{\max\{1, \tilde{\mathbf{p}}^{n+1}\}}$$

Where the gradient is approximated with the discrete forward difference

$$\nabla \mathbf{u} = (u_{i+1,j} - u_{i,j}, u_{i,j+1} - u_{i,j})$$

and the divergence with the discrete backward difference (superscripts denoting the first and second components)

$$\textbf{div} \, \mathbf{p} = p^1_{i,j} - p^1_{i,j-1} + p^2_{i,j} - p^2_{i-1,j}$$

Given a fixed u the update step for v is given as

$$\mathbf{v} = \mathbf{u} + \left\{ \begin{array}{cc} \lambda \alpha \nabla I_1 & \text{if } \rho(\mathbf{u}) < -\lambda\alpha |\nabla I_1|^2 \\ -\lambda \alpha \nabla I_1 & \text{if } \rho(\mathbf{u}) > \lambda\alpha |\nabla I_1|^2 \\ \rho(\mathbf{u}) \nabla I_1 / |\nabla I_1|^2 & \text{if } \rho(\mathbf{u}) \geq -\lambda\alpha |\nabla I_1|^2 \end{array} \right.$$

Where $\rho = I_1(\mathbf{x} + u_0) + (u - u_0)\nabla I_1 - I_0$ is the current residual. On top of this alternative optimization, we implement a median filter on the flow field, and use image pyramids in a coarse-to-fine approach in order to handle larger scale motions. That is, flow is computed on coarse sub-sampled images, and then propagated as an initial estimate for the next pyramid level.

In both algorithms, there are several points at which an iterative solution for the flow vector of an individual pixel in the image can be computed concurrently with its neighbors. This allows us to improve performance from a serial implementation by using OpenCL. In particular, we wrote OpenCL kernels to compute flow updates, image gradients, and local flow averages for Horn-Schunck, as well as kernels for $\mathbf{u}$ and $\mathbf{v}$ updates, and discrete forward and backward differences.

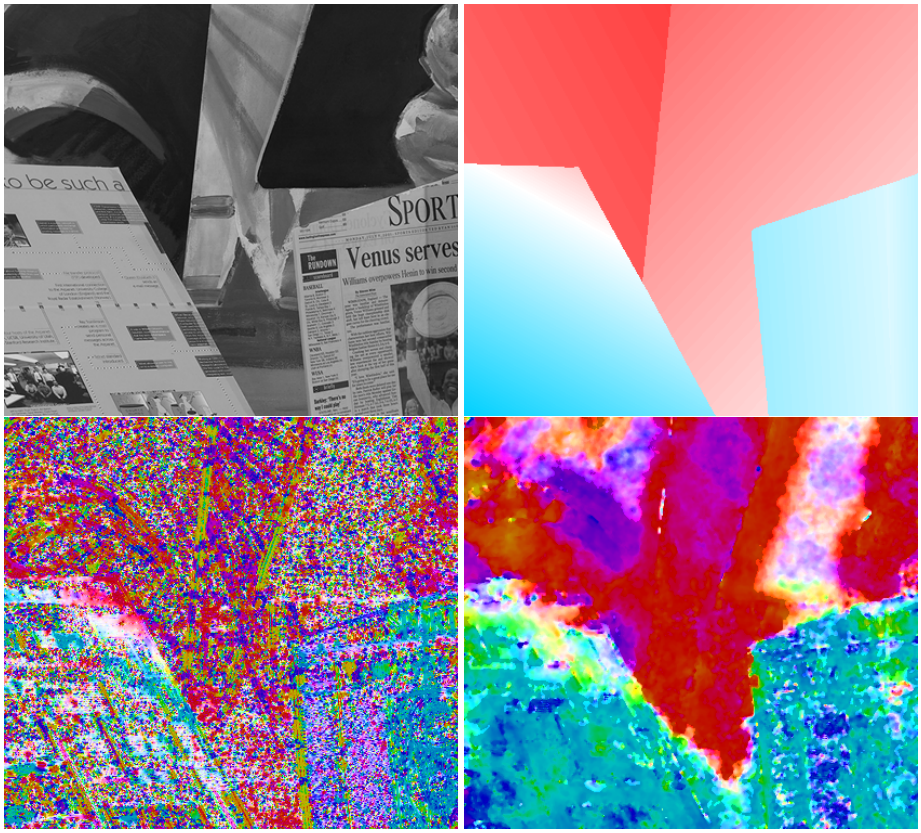

Figure 1: Comparison of flow fields calculated for the Venus dataset. Top row: Original image, ground truth flow. Bottom: Horn-Schunck, TV-L^1.

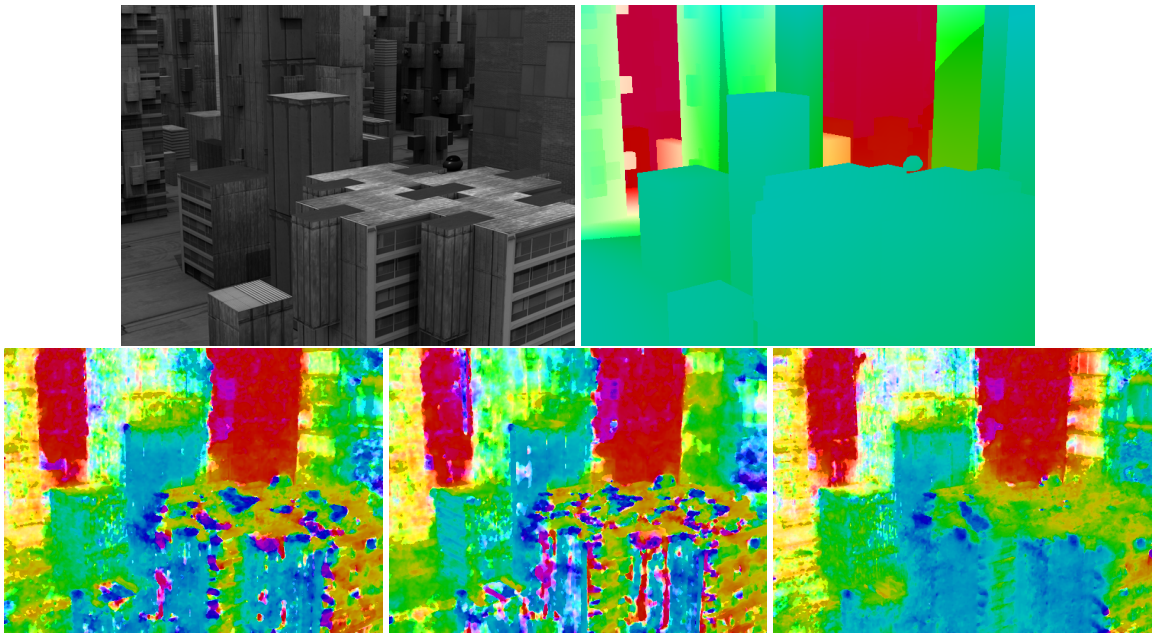

Figure 2: Comparison of different settings for pyramid levels and fixed point iterations. Top row: original image, ground truth flow. Bottom row: the output using the same settings in Figure 3, 4 pyramid levels and 14 iterations, and 14 levels and 4 iterations.

	Dimetrodon	Grove2	Hydrangea	RubberWhale	Urban2	Venus
Average HS	1.99	3.91	3.84	0.67	9.69	4.62
Average TV- L^1	1.43	1.79	1.97	0.69	7.33	2.58
Max HS	223.04	545.86	294.06	71.21	358.19	447.7
Max TV-L^1	4.3	9.55	16.01	6.02	27.57	11.83

Figure 3: Comparison of average endpoint error between Horn-Schunck and TV-L^1

4 Evaluation

We ran both the Horn-Schunck and TV-L^1 algorithms on a subset of the Middlebury dataset. Figure 3 lists average and maximum endpoint error in pixels for both algorithms for each dataset. For this table, we ran Horn-Schunck for 60 iterations. More than 60 iterations showed minimal improvement. TV-L^1 ran for 7 fixed point iterations, and used a pyramid of 7 images with a 0.9 scaling factor. Figure 1 gives an example of output flow compared to ground truth on the Venus dataset.

Figure 2 illustrates the effect of the number of fixed point iterations and the size of the image pyramid used.

We also compared runtimes of a reference Matlab implementation of TV-L^1, a straightforward port of the Matlab code using LibJacket, and our own hand written OpenCL code for the *RubberWhale* data set. We ran all three on the same machine, a MacBook Pro with an Intel Core i7 2.66GHz CPU and Nvidia GeForce GT 330M GPU, using 7 pyramid levels with a 0.66 scale factor and 5 fixed point iterations per level. We averaged the runtimes over 5 runs after an initial untimed "warm up" run. On the *RubberWhale* data set the reference Matlab implementation ran in an average of 8.7427 seconds, LibJacket took 0.7163 seconds, and our hand written OpenCL implementation 0.4026 seconds. On the *Venus* data set, our OpenCL implementation runs at 3fps.

5 Discussion

Figure 3 shows that TV-L^1 performs better than Horn-Schunck in almost all cases. The difference between the two methods is most clearly seen on the Urban2 setting, which has the largest apparent motion of the group. Since TV-L^1 makes use of an image pyramid, we are able to capture larger motion, so this make sense. Figure 3 also shows that the Horn-Schunck method produces larger single pixel error, which makes sense because the TV-L^1 method applies a median filter, and can throw out such outliers.

The parameters used in Figure 3 give roughly the same number of iterations to each method: Horn-Schunck uses a fixed 60 iterations, and TV-L^1 performs 7 fixed point iterations at each of 7 pyramid levels. By altering the ratio of fixed point iterations to pyramid levels, the performance of the algorithm on a particular dataset changes given the same "budget of iterations. By increasing the number of pyramid levels, the algorithm can capture larger motions, while increasing fixed point iterations improves the flow estimates at each pyramid level. Figure 2 shows this accuracy trade off. Performance wise, increasing the number of pyramid levels is more costly than increasing the number of inner iterations, as pyramids involve more memory transfer overhead.

Looking back at the entire project, one important lesson is that while the alternative formulation of the optimization problem leads to a more robust solution, probably the most important improvements the TV-L^1 approach has over standard Horn-Schunck are the inclusion of an image pyramid to handle larger scale motion, and the median filter to remove outliers.

The purpose of implementing these algorithms on the GPU was to achieve some performance gain over the standard serial version. Our first naive implementation of both TV-L^1 and Horn-Schunck algorithms made use of global memory for all computations, which could certainly be improved in future work. A good analogy is to compare working on global memory versus shared/texture memory in a GPU to operating on data directly on hard disk versus RAM/cache. Using global GPU memory allows for simpler programming of kernels, but is much slower. An obvious future work item would be to optimize the naive OpenCL kernels and make use of shared/texture memory. Even with this simple approach, we saw over an order of magnitude improvement in the running time from the reference Matlab implementation.

References

[1] CUDA C Programming Guide, May 2011.

[2] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. Black, and R. Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92:1–31, 2011. 10.1007/s11263-010-0390-2.

[3] B. Horn and B. Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

[4] B. Lucas, T. Kanade, et al. An iterative image registration technique with an application to stereo vision. In *International joint conference on artificial intelligence*, volume 3, pages 674–679. Citeseer, 1981.

[5] Y. Mizukami and K. Tadamura. Optical flow computation on compute unified device architecture. In *Image Analysis and Processing,14th International Conference on*, pages 179–184. IEEE, 2007.

[6] A. Wedel, T. Pock, C. Zach, H. Bischof, and D. Cremers. An improved algorithm for tv-l 1 optical flow. *Statistical and Geometrical Approaches to Visual Motion Analysis*, pages 23–45, 2009.