Practical Course GPU Programming
Project Assignment
winter semester 2012/13

Martin Oswald, Frank Steinbruecker

March 15, 2013

1 Assignment

Your assignment in this project is to implement a variational motion estimation / optic flow method using the NVIDIA CUDA Framework.

Comment your code well and hand it in at the end of your project.

Prepare a 20-25min presentation about your work explaining the benefit of parallel computation for the methods you have implemented and display the runtime results of your method. The presentation should include a short live demonstration of your program as well.

2 Theoretical Details

The general idea of optic flow is that given two images $I_1, I_2 : \Omega \to \mathbb{R}$, one computes a vector field $u : \Omega \to \mathbb{R}^2$, that matches the image intensities:

$$I_2(\mathbf{x} + u(\mathbf{x})) = I_1(\mathbf{x})$$

Naturally, for an intensity value in a pixel in I_1 there may be many pixels in I_2 that have the same value. Also, there might be none at all. Therefore, it is common to formulate the problem as a variational approach with a data term penalizing deviations

from the intensity values and a regularity term penalizing spatial variations of the flow field:

$$E(u) = \int_\Omega (I_2(\mathbf{x} + u(\mathbf{x})) - I_1(\mathbf{x}))^2 + \lambda |\nabla u|^2 \quad \mathrm{d}\mathbf{x}$$

For the two-dimensional flow field $u(\mathbf{x}) = [u_1(\mathbf{x}), u_2(\mathbf{x})]^\top$, the gradient magnitude $|\nabla u|$ is defined as

$$|\nabla u| = \sqrt{{u_1}_x^2 + {u_1}_y^2 + {u_2}_x^2 + {u_2}_y^2}$$

where the subindices $_x$ and $_y$ indicate derivatives in x and y direction.
The minimum of this energy is a flow field that is smooth and matches the image intensities reasonably well.
The equation above however is highly non-convex, since the desired argument u is an argument of the image intensities I. Therefore, a typical approach is to approximate $I_2(\mathbf{x} + u(\mathbf{x}))$ by a first order Taylor expansion. For this, I_1 is regarded as an image at time t, and I_2 is regarded as an image at time $t + 1$.:

$$I_2(\mathbf{x} + u(\mathbf{x})) = I(\mathbf{x} + u(\mathbf{x}), t + 1) \approx I(\mathbf{x}, t) + \nabla I^\top u + \underbrace{I_t}_{\frac{\mathrm{d}I}{\mathrm{d}t}} \tag{1}$$

Concerning the implementation, the temporal derivative I_t is simply the difference $I_2 - I_1$ at each pixel, while the gradient ∇I is usually averaged over both images for each pixel, since it is arbitrary whether one chooses to linearize the spatial displacement at time t or at time $t + 1$. Plugging this into the equation, we get

$$\begin{aligned} E(u) &= \int_\Omega (I(\mathbf{x}, t) + \nabla I^\top u + I_t - I(\mathbf{x}, t))^2 + \lambda |\nabla u|^2 \quad \mathrm{d}\mathbf{x} \\ &= \int_\Omega (\nabla I^\top u + I_t)^2 + \lambda |\nabla u|^2 \quad \mathrm{d}\mathbf{x} \end{aligned} \tag{2}$$

This is the method of Horn, B. and Schunck, B. 1981. “Determining optical flow.” in ”Artificial Intelligence“, 17:185-203.(You do not have to read this)

For the two-dimensional flow field $u(\mathbf{x}) = [u_1(\mathbf{x}), u_2(\mathbf{x})]^\top$, one gets two Euler-Lagrange Equations:

$$\begin{aligned} 0 &= L_{u_1} - \frac{\partial}{\partial x} L_{{u_1}_x} - \frac{\partial}{\partial y} L_{{u_1}_y} \\ 0 &= L_{u_2} - \frac{\partial}{\partial x} L_{{u_2}_x} - \frac{\partial}{\partial y} L_{{u_2}_y} \end{aligned}$$

where the subindices stand for the partial derivative of the Lagrangian integrand with respect to the partial derivative of u. In the case of the Horn and Schunck equation

the terms read the following way:

$$\begin{aligned} L_{u_1} &= 2(I_x^2 u_1 + I_x I_y u_2 + I_x I_t) \\ L_{u_2} &= 2(I_x I_y u_1 + I_y^2 u_2 + I_x I_t) \\ L_{u_{1_x}} &= 2\lambda u_{1x} \\ L_{u_{2_x}} &= 2\lambda u_{2x} \\ L_{u_{1_y}} &= 2\lambda u_{1y} \\ L_{u_{2_y}} &= 2\lambda u_{2y} \end{aligned}$$

The Euler-Lagrange equations then read

$$\begin{aligned} 0 &= I_x^2 u_1 + I_x I_y u_2 + I_x I_t - \lambda \Delta u_1 \\ 0 &= I_x I_y u_1 + I_y^2 u_2 + I_y I_t - \lambda \Delta u_2 \end{aligned}$$

Since 1981, there have been developed a large amount of improvements of this method, some of which you are going to implement as well. As a reference, read Papenberg et al. "Highly accurate optic flow computation with theoretically justified warping.". (**http://www.mia.uni-saarland.de/Publications/papenberg-ijcv06.pdf**) Except for advanced constancy assumptions and temporal smoothness of the flow field, your task is to implement the method presented in this paper, with the features listed below.

We advise you to build up your project in the following steps:

2.1 Original Approach

Implement the method of Horn and Schunck '81:

$$E(u) = \int_\Omega (\nabla I^\top u + I_t)^2 + \lambda |\nabla u|^2 \quad \mathrm{d}\mathbf{x}$$

with quadratic data and regularity terms resulting in linear Euler-Lagrange equations.

2.2 Advanced Penalty Functions

Implement the method with a robust regularity penalty function as you did previously for image regularization $\Phi(s^2) = \sqrt{\epsilon + s^2}$:

$$E(u) = \int_\Omega \Phi_D((\nabla I^\top u + I_t)^2) + \lambda \Phi_R(|\nabla u|^2) \quad \mathrm{d}\mathbf{x}$$

Different to the image regularization method you implemented in the course, here also the data term has a penalty function Φ_D that yields better robustness against noisy outliers.

2.3 Warping

Implement the method with the course-to-fine warping approach as described in the paper above.
Warping is somewhat similar to the fixed point iteration scheme you use for the last subsection. If you have a nonlinear diffusivity, you compute an approximate solution of an equation system with a fixed diffusivity, and then update the diffusivity with the computed solution. Warping means having an approximate solution u^k, and splitting the desired u into the known part u^k and the unknown increment du^k:

$$\begin{aligned}
E(u) &= \int_\Omega \Phi_D((I_2(\mathbf{x}+u(\mathbf{x}))-I_1(\mathbf{x}))^2)+\lambda\Phi_R(|\nabla u|)^2 \quad \mathrm{d}\mathbf{x} \qquad \Rightarrow \\
E(du^k) &= \int_\Omega \Phi_D((I_2(\mathbf{x}+u^k(\mathbf{x})+du^k(\mathbf{x}))-I_1(\mathbf{x}))^2)+\lambda\Phi_R(|\nabla u^k+\nabla du^k|^2) \quad \mathrm{d}\mathbf{x} \\
&\approx \int_\Omega \Phi_D((\nabla {I^k}^T du^k+\underbrace{I_2(\mathbf{x}+u^k(\mathbf{x}))-I_1(\mathbf{x})}_{I_t^k})^2)+\lambda\Phi_R(|\nabla u^k+\nabla du^k|^2) \quad \mathrm{d}\mathbf{x} \\
&= \int_\Omega \Phi_D((\nabla {I^k}^T du^k+I_t^k)^2)+\lambda\Phi_R(|\nabla u^k+\nabla du^k|^2) \quad \mathrm{d}\mathbf{x}
\end{aligned}$$

Practically, this means you create a warped image $I^k_{2\mathrm{warped}}$, by looking it up in I_2 with the computed flow u^k:

$$I^k_{2\mathrm{warped}}(\mathbf{x}) := I_2(\mathbf{x}+u^k(\mathbf{x}))$$

From this image, you create the new image gradient ∇I^k and temporal derivative I_t^k. If the warped second image matches the first image perfectly, the resulting incremental flow du^k will be zero, since the already computed flow u^k is perfect and $I_t^k \equiv 0$.
You proceed just as in equation (2), with the difference of having an additional right-hand-side expression $\lambda\Delta u^k$ in the Euler-Lagrange-Equations:

$$\begin{aligned}
0 &= \Phi'_D\cdot({I_x^k}^2 du_1^k + I_x^k I_y^k du_2^k + I_x^k I_t^k) - \lambda\mathrm{div}(\Phi'_R\cdot\nabla u_1^k) - \lambda\mathrm{div}(\Phi'_R\cdot\nabla du_1^k) \\
0 &= \Phi'_D\cdot(I_x^k I_y^k du_1^k + {I_y^k}^2 du_2^k + I_y^k I_t^k) - \lambda\mathrm{div}(\Phi'_R\cdot\nabla u_2^k) - \lambda\mathrm{div}(\Phi'_R\cdot\nabla du_2^k)
\end{aligned}$$

These are solved for du^k, and u^k is regarded constant. After each warping iteration, the increment is added to the known flow:

$$u^{k+1} := u^k + du^k$$

Although the warping strategy might improve the flow results on one image resolution alone, the really interesting fact is, that you can start with very small versions of the image, and compute the coarse, far-reaching flow on those scales, and then upsample the flow, and compute the incremental flow on the next finer scale. This way, you are able to compute the flow between two images that spans over several pixels and naturally violates the assumption of a small motion required for linearization as in (1).

2.4 CPU Implementation

We also ask you to implement the Warped Nonlinear version on the cpu and check differences concerning the runtime. This however will be regarded rather as a bonus for your project as well as for the presentation, since runtime comparisons are a nice thing to present.

3 Practical Details

Do not allocate or free any GPU memory during optical flow computation. The variables allocated and freed in the provided class are sufficient. Stick to our programming framework, either leave the function heads unaltered, or thoroughly explain what you are changing and why!

3.1 Discretization

3.1.1 Spatial Derivatives

For spatial derivatives, use central differences.

3.1.2 Temporal Derivatives

For temporal derivatives, use forward differences (which is somewhat intuitive, given the fact we are only computing the flow from one image to the next). This means

$$I_t(x, y) \approx I_2(x, y) - I_1(x, y)$$

which is coherent with the formulation in the last section.

3.1.3 Shared Memory

You will notice that we have declared shared memory of size $[SF\_BW][SF\_BH]$ in the kernels, and not $[SF\_BW + 2][SF\_BH + 2]$. This is due to the fact that all the boundary values are only needed by exactly one single thread, and therefore, there is no necessity to load them into shared memory. The check for block boundary positions can be moved from the load phase to the computation phase of the kernel, which simplifies the load phase ans complicates the computation phase. In the diffusion example this was the case as well, but it was a bit easier the way we did it.
Here however, we want you to do it the other way, since it saves shared memory and might speed up the computation, if more blocks can run in parallel.

3.2 Evaluation

For evaluation purposes, it is sufficient to load two consecutive images of a sequence, compute the optic flow from the first one to the second one, and compare it to the ground truth of that sequence. This can be done qualitatively by having a look at the color-code of the flow (have a look at our provided functions), or quantitatively, by computing the average end-point error defined as

$$AEE = \frac{\sum_{\text{pixels}(x,y)} |u_{\text{computed}}(x,y) - u_{\text{truth}}(x,y)|}{\sum_{\text{pixels}(x,y)} 1}$$

to some flow fields provided by us.

4 Behavioral Details

If you have questions, ask us! We prefer answering questions very much over being given intransparent and wrong GPU code. Also ask us if you have theoretical questions regarding Euler-Lagrange equations etc. (They can be a pain)
Your presentation will be graded mainly on its clarity and will yield 20% of your final grade. The success of your project will be graded separately with 50%. If you wish to collaborate with each other (pairs of two), tell us NOW. And comment your code.