GPU Programming in Computer Vision

CUDA Project Assignment

Evgeny Strekalovskiy, Jakob Engel, Julia Bergbauer

August 30, Summer Semester 2013

1 General Project Introduction

Your assignment in this project is to implement a variational motion estimation / optical flow method as well as a superresolution method using the NVIDIA CUDA Framework. Comment your code well.

The final version of your code has to be sent to `cuda-ss13@in.tum.de` in a zip archive **24 hours** prior to your presentation.

Prepare a 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 Organizational Details

You work in the assigned groups. The workload and the general understanding of the project should be equally distributed within the team.
If you have questions, send an email to `cuda-ss13@in.tum.de`.

2.1 Final Presentation

Please prepare a 20min presentation of your work.

2.1.1 General requirements

- the presentation should have a clear structure and slide layout
- give a general overview over the project and its goals
- explain how you achieved your goals
- give an overview over who did what in the project (coding, presentation, organization, etc.)
- discuss where you have encountered difficulties and how you solved them

2.1.2 Requirements on the technical and mathematical part

- explain the main ideas of optical flow and super-resolution estimation.
- emphasize on how various GPU programming techniques have been applied in the implementation and *why*.
- show and explain/discuss experimental results.
- give a live demo of your project (optical flow and super-resolution)

2.1.3 Further notes

- the presentation can be in English or German
- everybody in the group should present roughly for the same amount of time

2.2 Presentation Q&A

After your presentation there will be a 25min Q&A session, where we will ask you some questions about your work.

- you should be familiar with the CUDA programming model, for instance with the hierarchical memory model and thread organization
- you should have a good understanding of the both energies (optical flow and superresolution) and be able to explain the meaning of each term in each energy
- we might ask you about the general programming framework of the project
- we will not ask about the theoretical part of the optimization or any derivations

2.3 Assessment

- **30% basic implementations (first week)**: We will thoroughly evaluate your programming assignments of the first week.
- **20% final project presentation**: See above (2.1).
- **50% final project code quality / results; presentation Q&A**: That will be based on how well your program works. Further, we assess your answers on questions about the project and CUDA related topics. See above (2.2).

3 Theoretical Details

3.1 Optical Flow

3.1.1 Original Quadratic Approach

The general idea of optical 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 $\mathbf{x} = (x, y)$ in I_1 there may be many pixels $\mathbf{x} + u(\mathbf{x})$ 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:

$$E(u) = \int_\Omega \big(I_2(\mathbf{x} + u(\mathbf{x})) - I_1(\mathbf{x})\big)^2 \; + \; \lambda |(\nabla u)(\mathbf{x})|^2 \;\; \mathrm{d}\mathbf{x} \tag{1}$$

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

$$|(\nabla u)(\mathbf{x})| = \sqrt{(\partial_x u_1)^2 + (\partial_y u_1)^2 + (\partial_x u_2)^2 + (\partial_y u_2)^2}$$

The minimum of this energy is a flow field $u : \Omega \to \mathbb{R}^2$ that is smooth and matches the image intensities reasonably well.
The energy (1) above however is highly non-convex, since the desired argument u arises as 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 and I_2 are considered as instances of a time-dependant image $I(x, t)$ at two different times t and $t + 1$, respectively, i.e. $I_1(\mathbf{x}) = I(\mathbf{x}, t)$ and $I_2(\mathbf{x}) = I(\mathbf{x}, t + 1)$:

$$\begin{aligned} I_2(\mathbf{x} + u(\mathbf{x})) &= I(\mathbf{x} + u(\mathbf{x}), t + 1) \\ &\approx I(\mathbf{x}, t) + (\nabla I)(\mathbf{x}, t)^\top u(\mathbf{x}) + (\partial_t I)(\mathbf{x}, t) \end{aligned} \tag{2}$$

Here $(\nabla I)(\mathbf{x}, t) = [(\partial_x I)(\mathbf{x}, t), (\partial_y I)(\mathbf{x}, t)]^\top \in \mathbb{R}^2$ is the gradient w.r.t. the spatial variables. For brevity, we will write

$$\begin{gathered} I = I(\mathbf{x}, t), \quad \nabla I = (\nabla I)(\mathbf{x}, t), \quad I_x = (\partial_x I)(\mathbf{x}, t), \quad I_y = (\partial_y I)(\mathbf{x}, t), \\ I_t = (\partial_t I)(\mathbf{x}, t), \quad u = u(\mathbf{x}), \quad \nabla u = (\nabla u)(\mathbf{x}). \end{gathered}$$

Concerning the implementation, the temporal derivative I_t is simply the difference

$$I_t = I(\mathbf{x}, t + 1) - I(\mathbf{x}, t) = I_2(\mathbf{x}) - I_1(\mathbf{x})$$

at each pixel $\mathbf{x}$, while the gradient ∇I is usually averaged over both images for each pixel,

$$\nabla I \approx \frac{1}{2}\big((\nabla I_1)(\mathbf{x}) + (\nabla I_2)(\mathbf{x})\big)$$

since it is arbitrary whether one chooses to linearize the spatial displacement at time t or at time $t+1$. Plugging (2) into the original energy (1), we get the *linearized* energy

$$\begin{aligned} E(u) &= \int_\Omega \Big(I + \nabla I^\top u + \partial_t I - I\Big)^2 + \lambda |\nabla u|^2 \quad \mathrm{d}\mathbf{x} \\ &= \int_\Omega (\nabla I^\top u + I_t)^2 + \lambda |\nabla u|^2 \quad \mathrm{d}\mathbf{x} \end{aligned} \tag{3}$$

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 $u(\mathbf{x}) = [u_1(\mathbf{x}), u_2(\mathbf{x})]^\top \in \mathbb{R}^2$, one gets two Euler-Lagrange equations:

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

These equations must 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 the paper[1]

Papenberg et al., IJCV 2006
"Highly accurate optical flow computation
with theoretically justified warping.'.

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.

3.1.2 Advanced Penalty Functions

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

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

Differently than in 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.

3.1.3 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 which you use for the last subsection 3.1.2. In the fixed-point iteration for nonlinear diffusivity,

[1] `http://www.mia.uni-saarland.de/Publications/papenberg-ijcv06.pdf`

you compute an approximate solution of a system of equations by fixing the diffusivity, and then update the diffusivity using this computed solution. Warping means having an approximate solution u^k, and computing the desired u as the sum of known part u^k and an unknown increment du^k, i.e. $u^{k+1} = u^k + du^k$. In other words, we linearize the original energy (with a robust data term and gradient penalizers)

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

by using the Taylor expansion (2) at point $u = u^k$ instead of at $u = 0$. As the *linearized* energy we get

$$\begin{aligned} E(du^k) = & \int_\Omega \Phi_D\bigg(\Big(I_2\big(\mathbf{x} + u^k(\mathbf{x}) + du^k(\mathbf{x})\big) - I_1(\mathbf{x})\Big)^2\bigg) + \lambda\Phi_R\Big(|\nabla u^k + \nabla du^k|^2\Big) \quad \mathrm{d}\mathbf{x} \\ \approx & \int_\Omega \Phi_D\bigg(\Big(\nabla {I^k}^T du^k + I_2\big(\mathbf{x} + u^k(\mathbf{x})\big) - I_1(\mathbf{x})\Big)^2\bigg) + \lambda\Phi_R\Big(|\nabla u^k + \nabla du^k|^2\Big) \quad \mathrm{d}\mathbf{x} \\ = & \int_\Omega \Phi_D\bigg(\big(\nabla {I^k}^T du^k + I_t^k\big)^2\bigg) + \lambda\Phi_R\Big(|\nabla u^k + \nabla du^k|^2\Big) \quad \mathrm{d}\mathbf{x} \end{aligned} \tag{4}$$

Here we set

$$\nabla I^k = (\nabla I_2)\big(\mathbf{x} + u^k(\mathbf{x})\big), \quad I_t^k = I_2\big(\mathbf{x} + u^k(\mathbf{x})\big) - I_1(\mathbf{x}). \tag{5}$$

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 the temporal derivative I_t^k as in (5). 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 $I_t^k \equiv 0$.

In order to compute the minimizer du^k of the linearized energy (4), you proceed just as for (3) by computing the Euler-Lagrange equations, with the difference of having additional terms involving $\lambda\nabla u^k$ on the right-hand-side:

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

These are solved for du^k, and for this computation 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 (*downsampled* versions), 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 (2).

3.2 Superresolution

3.2.1 General Variational Superresolution Model

Consider an image $I : \Omega \to \mathbb{R}$, that has been degraded by a linear degrading operator F

$$I_D = FI.$$

(think of I as being a long vector with the number of component equal to the number of pixels, and of F as a matrix). The operator F is assumed to be singular, i.e. it eliminates certain parts of the image (e.g. when FI is a local average of I at each pixel). If we have *several* degraded observations of the image I, which arise by first transforming I by some linear operators T^i, $i \in \{1, ..., n\}$ and then applying the operator F, we get a large system of linear equations consisting of systems of linear equations for every observation i:

$$\begin{aligned} I_F^1 &= FT^1 I \\ I_F^2 &= FT^2 I \\ &\vdots \\ I_F^n &= FT^n I \end{aligned} \tag{6}$$

For every observation i *by itself*, the system $I_F^i = FT^i$ of linear equations is *under-determined* because F is singular. But the *combined* linear system of equations (6) is *in general overdetermined* (due to noise, for example). In other words, each pixel in the original image I is observed in multiple degraded images. This *redundancy* suggests that we can enhance the resolution of the degraded images, effectively reconstructing the high-resolution image I from the low-resolution ones I_F^i. However, it still may happen that the system of linear equations (6) it is under-determined.

Therefore, similar to the optical flow approach above, we penalize the deviations from a perfect solution, and also regularize:

$$E(I) = \sum_{i=1}^{n} \mu \left\| FT^i I - I_F^i \right\|_1 + \lambda \left\| \nabla I \right\|_1 \tag{7}$$

In case the system (6) is over-determined (i.e. they may not exist an exect solution I to (6)) this then chooses some solution which some the closest to *approximately* satisfy (6). And in case the system (6) is under-determined (i.e. they may exist infinitely many different solutions to (6)), the regularizer allows us to still obtain a unique solution. The regularizer introduces a *prior knowledge* into the problem, effectively it states that, given two candidate solutions which have equal values for the data term, one must choose the solution which is more regular.

3.2.2 Dualization

Up to now, the norm which we are using in the data term is the L_1-norm. For scalar of vector functions $f : \Omega \to \mathbb{R}^k$, $k \geq 1$, it is defined as

$$\|f\|_1 = \int_\Omega |f(\mathbf{x})| \mathrm{d}\mathbf{x} = \int_\Omega \Phi\big(|f(\mathbf{x})|^2\big) \mathrm{d}\mathbf{x}, \qquad \Phi(s) := \sqrt{s}$$

Since the computation of $(FT^iI)(\mathbf{x})$ in a single pixel $\mathbf{x}$ involves values of $I(\bar{\mathbf{x}})$ in *several* pixels $\bar{\mathbf{x}}$ (because F is singular, e.g. an averaging), the L_1-norm in (7) *couples* all variables in a non-trivial way, making the problem hard to solve. In the same way, the L_1-norm also couples many values of I also through the gradient in the regularizer part.

Therefore, what we do is to decouple the system by *dializing* the L_1-norm, in both the data term and the regularizer. This introduces *additional variables* intro the energy minimization, which are called *dual* variables. But both the data term and regularizer are become decoupled, in the sense that the expressions become *linear* in terms of the original image I. The key tool to dialyzing is the *convex conjugation*[2], and the fact that a convex function is equal to its convex *biconjugate*:

$$\|f\|_1 = \int_\Omega |f(\mathbf{x})| \, \mathrm{d}\mathbf{x} = \sup_{\xi, \|\xi\|_\infty \leq 1} \int_\Omega \langle f(\mathbf{x}), \xi(\mathbf{x}) \rangle \, \mathrm{d}\mathbf{x} \tag{8}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product. The dual variable $\xi : \Omega \to \mathbb{R}^k$ is defined at each pixel $\mathbf{x} \in \Omega$ and is a k-vector, it has the same number k of components at each pixel as the function $f : \Omega \to \mathbb{R}^k$ itself. For scalar-valued images, the scalar product is a simple multiplication. The constraint $\|\xi\|_\infty \leq 1$ means that $|\xi(\mathbf{x})| \leq 1$ must hold at each pixel $\mathbf{x} \in \Omega$. Using the replacement (8) also has the advantage that it circumvents the *non-differentiability* of the L_1-norm, since the right hand side of (8) is differentiable on both f and ξ which makes the optimization much easier. In the following, we write $\langle I, \xi \rangle$ for $\int_\Omega \langle I(\mathbf{x}), \xi(\mathbf{x}) \rangle \, \mathrm{d}\mathbf{x}$. Energy (7) now becomes

$$E(I) = \sum_{i=1}^{n} \sup_{q^i, \|q^i\|_\infty \leq 1} \mu \left\langle FT^iI - I_F^i, q^i \right\rangle + \sup_{\xi, \|\xi\|_\infty \leq 1} \lambda \langle \nabla I, \xi \rangle \tag{9}$$

In contrast to (7), this expression has additional dual variables $q^i : \Omega \to \mathbb{R}$ for every i and $\xi : \Omega \to \mathbb{R}^2$. Note that we can push the scalar factors μ and λ into the definition of the solution set for the dual variables:

$$E(I) = \sum_{i=1}^{n} \sup_{q^i, \|q^i\|_\infty \leq \mu} \left\langle FT^iI - I_F^i, q^i \right\rangle + \sup_{\xi, \|\xi\|_\infty \leq \lambda} \langle \nabla I, \xi \rangle \tag{10}$$

For arbitrary operators such as F however, one can also put μ in the operator, i.e. one can reformulate the whole problem with $\bar{F} := \mu F$ instead of F. In the CPU sample solution, we implemented a mixture of both versions, allowing you to draw a portion inside the operator, and a portion in the limit of the dual variables. In the rest of these instructions, we will stick to the later one.

3.2.3 Huber Norm

In addition, we use the Huber Norm

$$\begin{aligned} \|f\|_\epsilon &= \int_\Omega |f(\mathbf{x})|_\epsilon \, \mathrm{d}\mathbf{x} \\ |I(\mathbf{x})|_\epsilon &:= \begin{cases} \frac{|f(\mathbf{x})|^2}{2\epsilon} & \text{if } |f(\mathbf{x})| \leq \epsilon \\ |I(\mathbf{x})| - \frac{\epsilon}{2} & \text{if } |f(\mathbf{x})| > \epsilon \end{cases} \end{aligned}$$

[2] `http://en.wikipedia.org/wiki/Convex_conjugate`

This smoothes out the non-differentiability of the L_1-norm at the zero. Note that as the Huber-parameter ϵ approaches 0, the Huber-norm approaches the L_1-norm. We mention this here and include this in the code to stay consistent with the paper, but in the CPU sample solution, ϵ is actually set to 0. The energy now reads as

$$E(I) = \sum_{i=1}^{n} \mu \left\| FT^i I - I_F^i \right\|_\epsilon + \lambda \left\| \nabla I \right\|_\epsilon \tag{11}$$

and the biconjugate formulation (you can check that, if you want)

$$E(I) = \sum_{i=1}^{n} \sup_{q^i, \|q^i\|_\infty \leq \mu} \left\langle FT^i I - I_F^i, q^i \right\rangle - \frac{\epsilon}{2\mu} \left\| q^i \right\|^2 + \sup_{\xi, \|\xi\|_\infty \leq \lambda} \langle \nabla I, \xi \rangle - \frac{\epsilon}{2\lambda} \left\| \xi \right\|^2 \tag{12}$$

3.2.4 Optimization

By (12) the energy minimization problem is now reformulated as a *saddle point problem*. The problem is convex in the *primal variable* I and concave in the dual variables q and ξ. The energy E is to be minimized w.r.t. the primal variables, and maximized w.r.t. the dual variables. This can be optimized by computing alternatingly a *gradient descent* step in I with fixed q and ξ, and a *gradient ascent* step in q and ξ with fixed I.
For this, note that we need the *adjoint* operators of F, T and ∇. The adjoint operator of the gradient ∇ is the negative divergence $-\text{div}$. The update steps look the following way:

Gradient ascent update step for the dual variables q (the ones which arise in the data term dualization):

$$\begin{aligned}
\left(q^i\right)^{k+1/3} &= \left(q^i\right)^k + \tau_d (FT^i \bar{I}^k - I_D^i) \\
\left(q^i\right)^{k+2/3} &= \underset{p}{\text{argmin}} \left\{ \frac{\left\| p - \left(q^i\right)^{k+1/3} \right\|^2}{2\tau_d} + \frac{\epsilon}{2\mu} |p|^2 \right\} \\
&= \frac{\left(q^i\right)^k + \tau_d (FT^i \bar{I}^k - {I_D^i}^k)}{1 + \frac{\epsilon \tau_d}{\mu}} \qquad \text{(feel free to check this)} \\
\left(q^i\right)^{k+1} &= \frac{\left(q^i\right)^{k+2/3}}{\max\left\{1, \frac{\left|\left(q^i\right)^{k+2/3}\right|}{\mu}\right\}} \qquad \text{(clipping to the feasible set)}
\end{aligned} \tag{13}$$

We use the notation $(q_i)^k$ to denote the value of the dual variable q^i at iteration k. Notation $(q_i)^{k+1/3}$ and $(q_i)^{k+2/3}$ means values (q_i) at two intermediate auxiliary steps, before computing the values $(q_i)^{k+1}$ at the next iteration. Please note that these update steps for the dual q are performed using the over-relaxed version $\bar{I}^k$ of I^k (defined below in (16)).

Gradient ascent update step for the dual variables ξ (the ones which arise in the regularizer dualization):

$$\begin{aligned}
\xi^{k+1/3} &= \xi^k + \tau_d \nabla \bar{I}^k \\
\xi^{k+2/3} &= \underset{p}{\operatorname{argmin}} \left\{ \frac{\left\| p - \xi^{k+1/3} \right\|^2}{2\tau_d} + \frac{\epsilon}{2\lambda} |p|^2 \right\} \\
&= \frac{\xi^k + \tau_d \nabla \bar{I}^k}{1 + \frac{\epsilon \tau_d}{\lambda}} \\
\xi^{k+1} &= \frac{\xi^{k+2/3}}{\max\left\{1, \frac{|\xi^{k+2/3}|}{\lambda}\right\}} \qquad \text{(clipping to the feasible set)} \qquad (14)
\end{aligned}$$

The notation ξ^k means the value of the dual variable ξ at iteration k. The values at two intermediate auxiliary steps are denoted by $\xi^{k+1/3}$ and $\xi^{k+2/3}$, which are used to compute the value ξ^{k+1} at the actual next iteration $k+1$.

Descent update of the primal variable:

$$I^{k+1} = I^k - \tau_p \left(-\text{div} + \sum_{i=1}^{n} {T^i}^\top F^\top (q^i)^{k+1} \right) \qquad (15)$$

The operator ${T^i}^\top$ is the adjoint operator[3] to T^i, and $F^\top$ the adjoint operator to F.

Over-relaxation of the primal variable for the next update of the dual variables:

$$\bar{I}^{k+1} = \omega I^{k+1} + (1-\omega) I^k, \qquad \omega \in [1, 2] \qquad (16)$$

This optimization scheme has the free parameters τ_p, the time-step size of the primal update, τ_d, the time-step size of the dual update, and ω, the over-relaxation parameter. Use the ones provided in the CPU version. Pay attention to the fact that the dual updates only rely on the over-relaxed version $\bar{I}$ of I.

Though this scheme might seem complicated at first glance, it is quite straightforward if you have a CPU reference implementation.
The remaining question is, of course, what the degrading operator F and the transformation operators T^i look like, and how we implement their adjoint (transposed) versions for (15).

3.2.5 The Superresolution Model you are supposed to implement on the GPU

We want you to implement the superresolution model of

Unger et al., DAGM 2010,
"A Convex Approach for Variational Super-Resolution.'.

on the GPU. This paper is provided as part of the material on our website.

[3] `http://en.wikipedia.org/wiki/Adjoint_operator`

Transformations T^i. In the case of this paper, the transformation operators T^i simply describe the backward warping performed by a given optical flow field. This means, if the flow u_{ij} describes the flow from image I_i to image I_j, i.e.

$$I_j(\mathbf{x} + u_{ij}(\mathbf{x})) = I_i(\mathbf{x})$$

this yields a warping operator W_{ij}:

$$(W_{ij}I_j) = I_i, \quad (W_{ij}I_j)(\mathbf{x}) := I_j(\mathbf{x} + u_{ij}(\mathbf{x}))$$

The adjoint warping operator is a bit tricky. It is basically the bilinear backward warping reversed (in the paper, the authors use a bicubic interpolation, but we use bilinear interpolation). In the *backward warping*, at a pixel position $\mathbf{x}$ you add the flow at that position and get $(\mathbf{x}+u(\mathbf{x}))$, bilinearly interpolate the image value $I_2(\mathbf{x} + u(\mathbf{x}))$ and, after computing the interpolations in all pixels, write the result image back to I_2 at $\mathbf{x}$. For the adjoint warping operator $W_{ij}^\top$ you perform a *forward warping*. This means, you take the image value at $I_1(\mathbf{x})$ and distribute that value in $I_2(\mathbf{x}+u(\mathbf{x}))$ to the four adjacent pixels belonging to the subpixel position $(\mathbf{x} + u(\mathbf{x}))$ with their corresponding coefficients according to bilinear interpolation. You accumulate all those values in the warped image. On the GPU, you have to figure out how to avoid race conditions in this somewhat random incrementation of image values.

Degrading operator F. The degrading operator F consists in our iteration of first a Gaussian blur operator B and a subsequent downsampling operator D. Concerning the blur operator, we want a different handling of the image boundaries in this case. We want to *mirror* the values outside the image boundary to values inside the image. To give an example, if we have to access the image value $I(-2, y)$ somewhere in the convolution, we want to map it to the position $I(1, y)$, and on the right side of the image domain, $I(w+1, y)$ maps to $I(w-2, y)$ on the other side, $I(w+2, y)$ maps to $I(w-3, y)$ etc. This way, the convolution preserves the average gray value of the image. A simple clamping of the image values to the boundaries does not preserve the average gray value. Mirroring the boundaries also makes the blur operator self-adjoint, i.e. $B^\top = B$.

The downsampling operator D is not really a downsampling, because you do not take samples of a function, but rather do a weighted average of the image. For the details we refer to the paper.

The energy now reads as

$$E(I^i) = \sum_{j \in N_i} \sup_{q^j, \|q^j\|_\infty \leq \mu} \left\langle DBW_{ji}I^i - I_F^j, q^j \right\rangle - \frac{\epsilon}{2\mu} \left\|q^j\right\|^2 + \sup_{\xi, \|\xi\|_\infty \leq \lambda} \left\langle \nabla I^i, \xi \right\rangle - \frac{\epsilon}{2\lambda} \|\xi\|^2 \tag{17}$$

This means the following:
You have been given a sequence of degraded images $I_F^i, i \in \{1, ..., n\}$, and you compute a higher-resolution image I^i for every i *separately* (one after another).

For every image I^i you also have the optical flows u_{ji} from other images I_F^j to that image, which are computed on the *higher resolution versions* of these images (obtained by trivial upsampling[4]) in a certain neighborhood of that

[4]Note that only I_F^j needs to be upsampled, since in the energy I^i is already assumed to be a high-resolution image.

image, lets say, 8 images before and after image i in the sequence. This means N_i would be $\{i-8,\dots,i+8\}$ for general i, so that $|N_i| = 17$, and e.g. for $i = 3$ one would take $N_i = \{1, 2, \dots, 17\}$.

The energy assumes the following image formation model: for each $j \in N_i$, the degraded image I_F^j has been produced by backward warping the higher resolution image I^i along the flow u_{ji} from j to i, and afterwards by blurring and downsampling that image ($I_F^j = DBW_{ji}I^i$). Penalizing deviations from this model yields the data term of the energy. The regularity term is usually very weakly weighted, i.e. $\lambda > 0$ is small. This is because what one actually wants to recover are the high-frequency irregularities of the image. The main purpose of the regularizer is to make the problem *well-posed*, i.e. to ensure that the solution is unique.

4 Practical Details

4.1 Framework

We provide you with a framework with a complete CPU version of both the optical flow computation and the superresolution computation. We documented the interface but not the computation code. In our sample solution, the CPU and GPU versions produce the same results. You might deviate from this a bit, for example, if you actually use the linear interpolation hardware from GPU textures, the floating point accuracy might differ from the CPU. However, as you can see, we also perform Red-Black-SOR on the CPU, which is somewhat counterintuitive, but ensures results equal to the ones on the GPU.

You have to implement the GPU functions corresponding to the linear operators in the superresolution formulation, the superresolution updates, and the optical flow updates. You also need the resampling (upscaling and downscaling) and blurring operator for optical flow computation, so we suggest you first focus on those. Second, focus on the optical flow computation, since you need the results of the optical flow method as input for the superresolution method. We provided a main file for testing the optical flow and another one for testing the superresolution against the CPU versions.

4.2 Discretization

4.2.1 Spatial Derivatives

In the optical flow part, we use the SOR method with central differences, while for the superresolution we use forward differences for the image gradients and backward differences for the divergence of the dual variable ξ. To produce correct adjoint operators (∇ and $-\text{div}$) with a vanishing image gradient at the boundary, the dual variable ξ has to be 0 outside the image. You can easily

verify this for yourself in the one-dimensional case, i.e.

$$\langle \nabla I, \xi \rangle = \int_\Omega \frac{\partial}{\partial x} I(x) \cdot \xi(x) \mathrm{d}x \approx \sum_{x=1}^{N} (I(x+1) - I(x)) \cdot \xi(x) = \sum_{x=1}^{N} I(x) \cdot \Big(-(\xi(x) - \xi(x-1)) \Big)$$

with $\xi(0) = 0$ and $I(N+1) = I(N)$. This is also closely related to integration by parts theorem.

4.2.2 Temporal Derivatives

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

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