Chair for System Simulation

Winter Term 2022/2023

Prof. Dr. Harald Köstler, Nils Kohl

Advanced Programming Techniques
Sheet 2 — Project: Optical Flow

This sheet contains the instructions on the semester project. The project is not mandatory. However, we strongly recommend that you work on it as it serves as a preparation for the exam. Successful completion is rewarded with bonus points in the exam.

Optical flow

The task is to write a program that computes the optical flow given an image sequence using the Horn–Schunck method [3]. Optical flow describes the apparent motion of objects that are displayed on the present image sequence. Successful computation yields a vector field that approximates the velocity of moving parts.

Figures 1a and 1b show an example sequence of two images that contain several moving rectangles. The computed optical flow field is visualized in fig. 1e. Grayscale images of the u and v components of the vector field are shown in figs. 1c and 1d.

Mathematical model

The derivation below skips many details that are described in more depth in [3, 2, 4].

Let $\Omega \subset \mathbb{R}^2$ be the image domain and

$$I(x, y, t) : \Omega \times [0, T] \to \mathbb{R} \tag{1}$$

a function that maps every point of the domain to a gray scale value for each point in time. The optical flow field $F(x, y, t) = (u(x, y, t), v(x, y, t))^\top$ can be approximated using the Horn-Schunck method as the minimizing argument of the energy functional

$$E = \iint \left((I_x u + I_y v + I_t)^2 + \alpha \left(|\nabla u|^2 + |\nabla v|^2 \right) \right) dy\, dx \tag{2}$$

where $\alpha \in \mathbb{R}$ is a weighting factor, and I_x, I_y, and I_t are the derivatives

$$I_x := \frac{\partial I}{\partial x},\ I_y := \frac{\partial I}{\partial y},\ I_t := \frac{\partial I}{\partial t}. \tag{3}$$

Using the Euler-Lagrange equations, the minimization problem is equivalent to the solution of the system of partial differential equations

$$\begin{aligned} \alpha \Delta u - I_x \left(I_x u + I_y v + I_t \right) &= 0, \\ \alpha \Delta v - I_y \left(I_x u + I_y v + I_t \right) &= 0. \end{aligned} \tag{4}$$

with homogeneous boundary conditions.

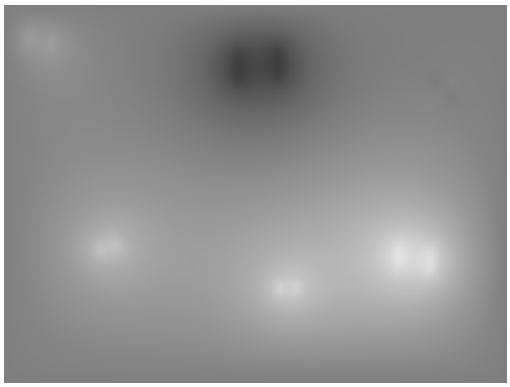
Discretization

The described method shall be used to compute the optical flow field of a sequence of two gray scale images with a bit-depth of 256. The resulting flow field is eventually also quantized to the same bit-depth. Details on the input and output image formats can be found later below.

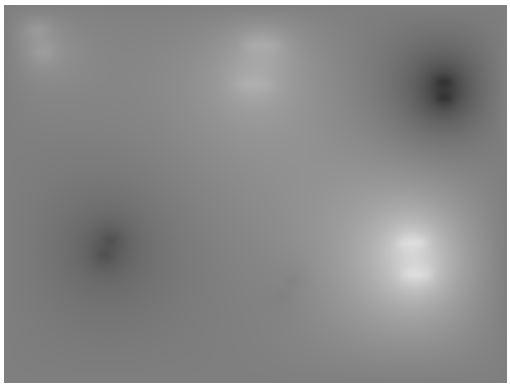
A finite difference discretization is applied to compute the derivatives. The image pixels are defined on the domain $\tilde{\Omega} = \{1, \ldots, N_x\} \times \{1, \ldots, N_y\}$. The number of pixels is denoted by $N = N_x N_y$. Using the

(a) I^t

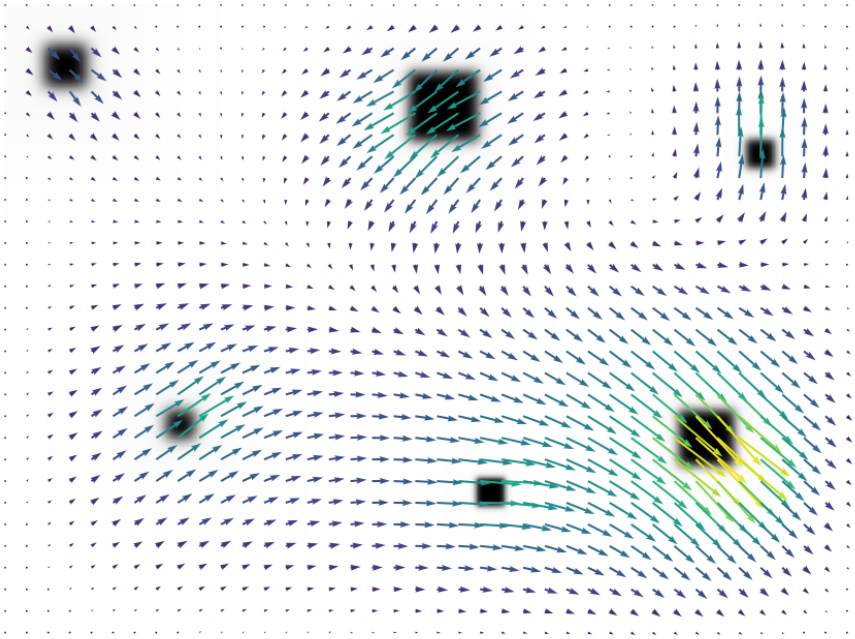
(b) I^{t+1}



(c) u



(d) v



(e) flow field constructed from u and v

Figure 1: Optical flow of several moving rectangles.

notation $I(i,j,t) = I^t_{i,j}$ to refer to value of the image at pixel (i,j) and time t the partial derivatives are approximated by

$$\begin{aligned} I_x &\approx \frac{1}{4}\left(I^t_{i,j+1} - I^t_{i,j} + I^t_{i+1,j+1} - I^t_{i+1,j} + I^{t+1}_{i,j+1} - I^{t+1}_{i,j} + I^{t+1}_{i+1,j+1} - I^{t+1}_{i+1,j}\right), \\ I_y &\approx \frac{1}{4}\left(I^t_{i+1,j} - I^t_{i,j} + I^t_{i+1,j+1} - I^t_{i,j+1} + I^{t+1}_{i+1,j} - I^{t+1}_{i,j} + I^{t+1}_{i+1,j+1} - I^{t+1}_{i,j+1}\right), \\ I_t &\approx \frac{1}{4}\left(I^{t+1}_{i,j} - I^t_{i,j} + I^{t+1}_{i,j+1} - I^t_{i,j+1} + I^{t+1}_{i+1,j} - I^t_{i+1,j} + I^{t+1}_{i+1,j+1} - I^t_{i+1,j+1}\right). \end{aligned} \tag{5}$$

The negative Laplace operator is approximated by the 5-point stencil

$$-\Delta u \approx \begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix}. \tag{6}$$

The discrete version of equation eq. (4) is thus

$$\begin{aligned} \left(I_x^2 + 4\alpha\right) u_{i,j} - \alpha\left(u_{i,j+1} + u_{i-1,j} + u_{i+1,j} + u_{i,j-1}\right) + I_x I_y v + I_x I_t = 0 \\ \left(I_y^2 + 4\alpha\right) v_{i,j} - \alpha\left(v_{i,j+1} + v_{i-1,j} + v_{i+1,j} + v_{i,j-1}\right) + I_x I_y u + I_y I_t = 0. \end{aligned} \tag{7}$$

This can be rewritten as the linear system of equations

$$Ax = b \tag{8}$$

where $A \in \mathbb{R}^{2N\times 2N}$ and $b \in \mathbb{R}^{2N}$ are computed from the input image sequence and $x = (u_1, \ldots, u_N, v_1, \ldots v_N)^\top \in \mathbb{R}^{2N}$ is the solution vector containing the discrete flow field.

Solving the linear system

While we prescribe the mathematical model and discretization that shall be used to compute an approximation to the optical flow we do not force you to use any specific linear solver. However, for the specific structure of the linear system, multigrid methods are among the fastest existing solvers and are therefore recommended. We refer to [5, 1] for a general introduction and details on multigrid.

Implementation

The task is to submit an implementation of the Horn-Schunck method that given several pairs of input images computes the corresponding optical flow field. **The task is not to implement the "*best*" method, but to implement the *described* method!** While we advise you to use C++ as the language of choice, we cannot and will not force you to do so. However, we will not give any advice in the exercises if you use anything else that plain C++. This project is the perfect opportunity to prepare for the exam.

Image formats All given images and those that shall be output are in grayscale bitmap (`.bmp`) format with a bit-depth of 256. As stated above, the task is to compute the optical flow given 2 images. We suggest that you

1. read in the `.bmp` images,
2. rescale the pixel values to the range $[0,1]$ and use a floating-point data type for the internal representation and all computations,
3. compute the result vectors u and v,
4. normalize the flow field so that the largest vector (u_i, v_i) has magnitude 1 (all entries u_i and v_i should now be in $[-1,1]$),

5. convert the result back to pixel values with a bit depth of 256 and quantize - this could be done as follows: given an entry $w \in [-1, 1]$ after normalization

$$w_{\texttt{quant}} = \texttt{clamp}(\texttt{(unsigned char)}(255 * (0.5 * (w + 1))),\ 0,\ 255) \tag{9}$$

yields the quantized pixel value.

The output, i.e. the optical flow field, shall be written to two output images that have the same size as the input images. The x and y components of the field F, that means the vectors u and v, shall thus be provided as an output.

For reference: the vector $(\tilde{u}, \tilde{v}) = (255, 255)$ points to the bottom right (compare the component images with the flow field in fig. 1 (0 is black, 255 is white)).

Workflow and submission This is a **group project**. We ask you to form groups of three students. Since that is not always possible, you can also form groups of two or do the project alone. However, if everyone submits their own solution, the evaluation will be more involved and might take longer. Use this opportunity to practice working on a software project as a team.

The structure of your submission is mostly up to you. You have the opportunity to learn to structure your project yourself. There are only a few requirements, listed below:

- You are forced to use our GitLab servers at the LSS to upload your submissions (if you want your submissions to be evaluated by us). We will provide you with an account and a repository where you simply commit and push your progress via `git` (details below). In regular intervals we will clone the project's master branch as is and perform the evaluation on it.
- You have to provide two shell scripts in the top-level directory of your repository:
  - `build.sh` is a shell script that, when executed, compiles and links your executable. This script would for instance run `cmake`, `make`, or directly invoke a compiler. It is up to you to put everything required to prepare your program in this script.
  - `run.sh` is a shell script that takes 4 positional arguments:

    ```
    $ run.sh <img_in_0> <img_in_1> <img_out_u> <img_out_v>
    ```

    `<img_in_0>` and `<img_in_1>` are two input images the optical flow field of which shall be computed. `<img_out_u>` and `<img_out_v>` are paths that specify where the output flow field components u and v shall be written to.

Evaluation To evaluate the correctness of your implementation your results are compared to those computed by a reference implementation. You will not have access to the reference code, but you are provided reference solutions for a set of input image sequences. Once you verified your implementation using the provided test images, we will evaluate it using additional test images that you will not have access to. The evaluation procedure has some error thresholds since a one-to-one correspondence is unlikely. Additionally, the performance of your implementation will be evaluated by measuring the required run time to compute the flow field of several input image sequences. The run time of your program will be compared to that of our reference implementation on a reference system. There will be some leeway regarding the thresholds and we will update you on the concrete numbers. **For the evaluation we assume $\alpha = 1$.**

The results of the evaluation will be accessible via GitLab[1]. At least once a week (but likely more frequently) all codes are evaluated, and the results are pushed to a directory in the repository. There are two files: `overview.txt` lists an overview of the results. `data.json` contains all detailed data including for example stdout and stderr of your `build.sh` and `run.sh` scripts.

[1] `https://i10git.cs.fau.de/advpt-student/advpt-opticalflow-evaluation-results`

Until otherwise specified, the evaluation is performed on a machine with an Intel Westmere processor (dual socket with 4 cores per socket) and 24 GB of main memory. To pass the numerical test stage, the computed solutions u, v, interpreted as vectors must fulfill

$$\frac{\|u - u^*\|}{\|u^*\|} \leq \varepsilon, \quad \frac{\|v - v^*\|}{\|v^*\|} \leq \varepsilon, \tag{10}$$

for each test case, where u^* and v^* are the reference solutions and $\|\cdot\|$ is one of $\|\cdot\|_2$ and $\|\cdot\|_\infty$. For now we choose $\varepsilon = 0.05$, i.e. we apply a 5% relative error threshold.

To pass the second stage, the run time of the `run.sh` script for each individual test is compared to the run time of the reference implementation. Let t be the run time of your implementation, and t^* the run time of the reference. We require

$$t/t^* \leq \tau, \tag{11}$$

where τ is set to 2 for now.

The parameters ε and τ may be adapted in the future.

Bonus points Bonus points on the exam are awarded in two steps. An equivalent of a grade improvement of 0.3 is awarded if your submission produces the correct results for all test images. If your solution produces the correct results and the measured run time is below the threshold, a total grade improvement of 0.7 is awarded.

Project registration Please form groups of three people. Registration will be open via StudOn.

Deadline for the registration is Friday, December 02 at 23:59.

There is a table where you can enter a group name and the three email addresses of the group members. **Please use your @fau.de email adresses.** Once you have registered, we will create a GitLab account for each of you on `https://i10git.cs.fau.de`. Additionally, we will create a repository containing the reference images and solutions for each group. You will be informed via mail. Please use the forum if you encounter any problems with the registration or your GitLab account.

Use the repository to write your implementation as described in this sheet. We will pull the `master` branch of your repositories on a regular basis and run our tests against it. The results will be published so you can observe whether your implementation works and is fast enough. Also we will upload `stdout` and `stderr` of the calls to `build.sh` and `run.sh` so you have the necessary information to fix your build process and program.

Please pay attention to the StudOn forum where we will post updates on the organization of this project.

Continuous integration Your repository will contain a directory with two pairs of test images and the respective reference solutions so that you can test your implementation. Additionally, it contains a file called `.gitlab-ci.yml` that triggers a continuous integration pipeline. Each time you push changes to your repository, the pipeline will be triggered and execute your build and run scripts. This way you can check whether your code builds properly on our system.

Reference system The reference system has `cmake` version 3.14.4 and `gcc` 11.1.0 installed.

Tips

- Read this sheet carefully!
- Study the referenced literature. Especially [3], since it describes the method you have to implement. Many details are omitted in the description above.

- We recommend to use the CImg library[2] to read and write images from and to file. The reference implementation uses CImg, too. It consists of a single header file that has to be included by your implementation. Images can be read and written as follows:

```
#include "CImg.h"

using namespace cimg_library;

// Read .bmp image from file.
CImg< unsigned char > img(filenameIn.c_str());

// Access single value of the first channel.
// You can assume that all channels of the provided input images
// contain the same value.
img(i, j) = /* ... */

// Write .bmp file.
img.save_bmp(filenameOut.c_str());
```

  By default, CImg requires X11 which is not supported on the test system. So you have to define `cimg_display` to be 0, before including the CImg header - for example like this:

```
#define cimg_display 0
#include "CImg.h"
...
```

- Test your image in- and output implementation. Does it really work as intended?
- Verify that you assemble all the vectors I_x, I_y, and I_t correctly. Think of ways to test your implementation.
- As a first step, it is sufficient to apply a simple numerical method to solve the linear system eq. (8). Don't start with multigrid right away. Move on once a simple solver computes the correct results.
- There are countless resources on `git` and version control in general online. For example this one `https://www.atlassian.com/git`.

References

[1] William L Briggs, Van Emden Henson, and Steve F McCormick. *A multigrid tutorial*. SIAM, 2000.

[2] Iris Christadler. *Mehrgitterverfahren für die berechnung des optischen flusses mit nicht-standardregularisierungen*. PhD thesis, Diploma thesis, University of Erlangen-Nuremberg, 2004.

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

[4] Harald Köstler. *A multigrid framework for variational approaches in medical image processing and computer vision*. University of Erlangen-Nuremberg, 2008.

[5] Ulrich Trottenberg, Cornelius W Oosterlee, and Anton Schuller. *Multigrid*. Elsevier, 2000.

[2] `https://cimg.eu/`