Lab on a Chip

TECHNICAL INNOVATION

Image processing and analysis system for the development and use of free flow electrophoresis chips – Supporting information

Sven Kochmann and Sergey N. Krylov

Table of contents

Additional files included

- Chip model files (Solid Edge)
- COMSOL simulation file (Version 5.1)
- Python FFE-library and programs including full API documentation

Chip fabrication

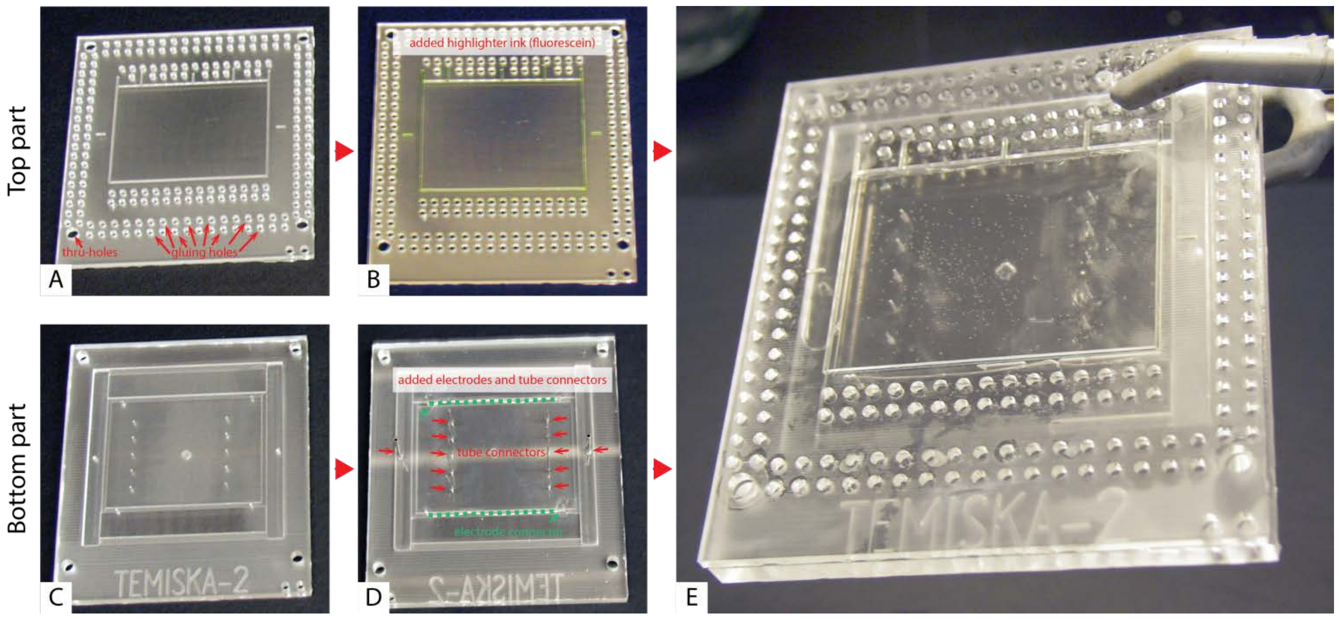

Figure S1. Assembly of the chip. Images show the top part of the chip without (A) and with (B) highlighter ink, the bottom part without (C) and with (D) added electrodes (green dashed lines) and tube connectors (red arrows) at inlets and outlets, and the final assembled chip (E). Please note that we increased the contrast of all pictures and the saturation of (B) to increase visibility.

The FFE chip was milled out of poly(methyl methacrylate) (PMMA, refractive index $n_D^{20°C}$ = 1.4918 [1]) using an MDX-540 milling machine (Roland DGA, Irvine, CA). A detailed description of the general milling procedure can be found elsewhere.[2]

The chip consists of a top and bottom part (see **Figure S1**). A marking scheme (1 mm deep channels) was carved in the top of the chip (**Figure S1A and B**) and coloured by a common yellow highlighter (0.7 mm wide). Common text markers consist of luminophores (in this case fluorescein), which are visible during imaging. Metal electrodes (platinum wire, 0.5 mm diameter) and metal tube connectors (Luer stub adapters, Gauge 16 and 25) were glued to the bottom part using common two-component epoxy glue. Both parts are aligned and temporarily fixed on top of each other by using bolts in the thru-holes (larger holes in the corner of both the top and bottom part). Subsequently, both parts are bonded together by introducing dichloromethane into the small gluing holes (only present on top part) to provide a strong and irreversible seal. After the bonding is complete (10–30 minutes), the bolts are removed.

The chip name (*Temiska-2*) was derived from *Temiskaming Shores*, a small town in the north of Ontario, Canada. The *2* indicates the serial number. FFE chip model was designed using Solid Edge (Siemens PLM, Plano, TX) software. Immersion oil (refractive index $n_D^{20°C}$ = 1.517) was applied to the top of the chip (the side with the markings) to improve the visibility and reduce scattering and refraction. Please note that we did not optimize the chip or the chip manufacture for this study. The chip possesses five sample inlets and five sample outlets.

Imaging chamber assembly

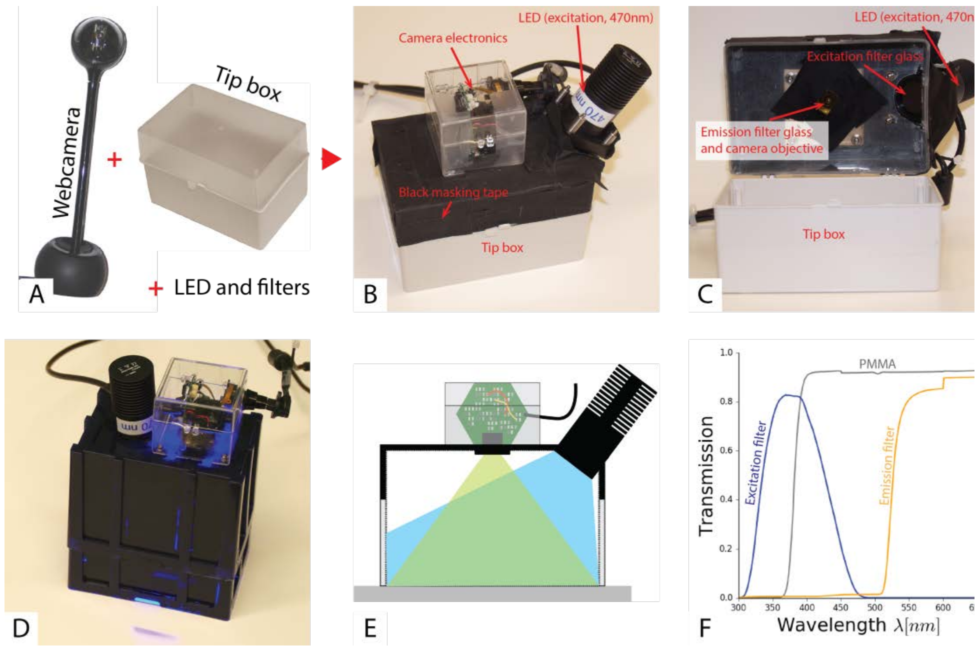

Figure S2. Assembly and components of image chamber. Images show the main components (A), the outside (B) and inside (C) of the imaging chamber, an example of another version of the imaging chamber (D), and a sketch of the imaging box (E). The plot in (F) shows the transmission spectra of PMMA, the excitation filter, and the emission filter. Filter and PMMA transmission spectra were recorded on a Beckman DU 520 Spectrophotometer (SCIEX, Vaughan, ON, Canada).

The image chamber was designed for photoluminescence imaging and consists of a camera objective, a micropipette tip box acting as a case, glass filters, and an LED for excitation (see **Figure S2**). A standard laptop (TravelMate 6592G from Acer) running Windows XP was used as the acquisition platform.

An empty micropipette tip box (127 × 85 × 82 mm) is acting as a case for the imaging chamber. It was adjusted by cutting in holes for fixing the camera electronics and the excitation light. Also, an imaging window/hole was cut into the bottom (100 × 70 mm). (Semi)-Transparent parts of the box were covered with black masking tape (see **Figure S2B**). Since the setup is very modular, it is simple to exchange parts to adapt to new situations. An example of another setup with a different case can be seen in **Figure S2D**.

The camera objective and electronics were extracted from a QuickCam® Orbit™ MP from Logitech (Newark, CA, USA). They were placed in a small plastic box prepared with holes for fixing and the USB cable. The box was fixed on top of the actual imaging chamber using small screws. The focus of the camera was adjusted with the focus ring on the objective. In front of the objective, an orange emission filter (see **Figure S2F** for spectrum) was placed.

A 470 nm LED (M470L3) from Thorlabs (Newton, NJ, USA) was used for illumination. An excitation filter was placed (see **Figure S2F** for spectrum) in front of the LED.

Signal distribution

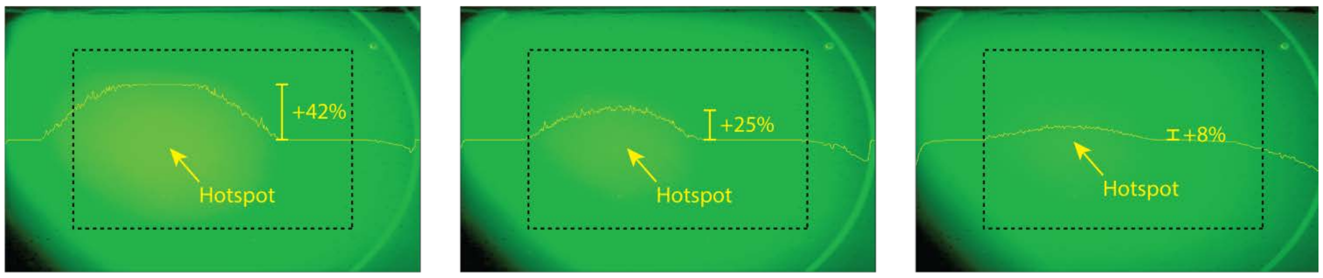

Figure S3. Three example pictures showing the signal distribution in the imaging chamber for different excitation intensities (arbitrary, decreasing from left to right). The yellow lines depict the light intensity in the centre of the frame (cross section; same scale for every picture). The black dashed rectangle shows the size of the separation zone of the FFE chip used in this study.

To check the signal distribution in the measurement area, we filled a large Petri dish with a fluorescein solution (500 µM) and placed the imaging system on top to take pictures at different excitation intensities. The three example pictures in **Figure S3** show that there is a hotspot on the left side of the image.

However, this hotspot can be tuned by adjusting the excitation light intensity. Here, reducing the intensity (from left to right picture) leads to an almost homogeneous signal distribution in the centre of the chamber with a large enough area to cover the separation zone of the FFE chip used in this study. Therefore, we did not use any additional lenses to shape the excitation or emission light for the presented study.

Experimental details

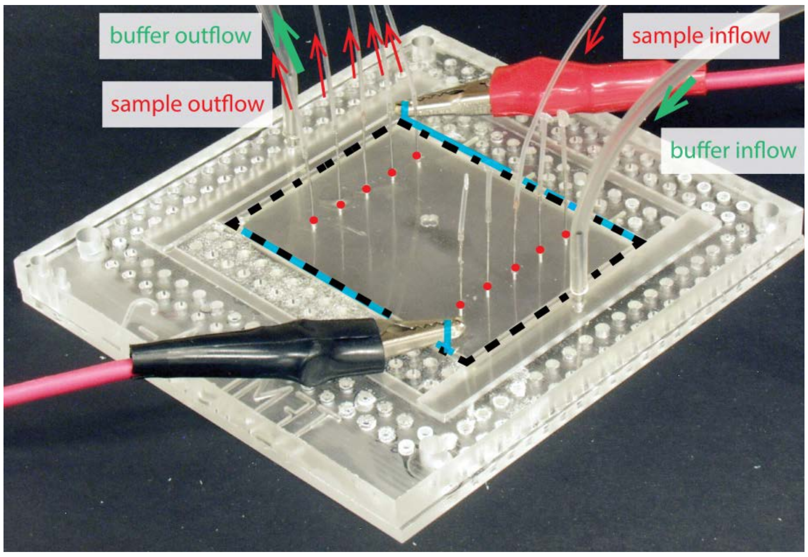

Figure S4. Bottom view of the chip showing the tube and electrode connections. Large tubing (3 mm inner diameter; polyethylene) is used for buffer in and outflow (green arrows), while small tubing (1/16" inner diameter; polyethylene) is used for sample in and outflow (red arrows and red circles). The black dashed lines mark the separation zone. The blue lines mark the electrode and the connectors at which the high voltage is applied (the electrodes themselves run over the whole side of the separation zone; see also **Figure S1D**).

Chemicals

All solutions were prepared using analytical grade reagents. 4-(2-hydroxyethyl)piperazine-1-ethanesulfonic acid sodium salt (HEPES), rhodamine 6G hydrochloride (rhodamine), fluorescein sodium salt (fluorescein), 5(6)-Carboxyfluorescein (carboxyfluorescein), sodium hydroxide, and immersion oil (UV-transparent, fluorescent-free) were purchased from Sigma Aldrich (Oakville, ON, Canada). Deionized water (18.2 MΩ cm^{-1}) was used for preparation of all solutions.

General experimental details

We used HEPES (10 mM, pH 7.6) as the BGE and solutions of rhodamine, fluorescein, and carboxyfluorescein (500 µM each) as *tracer dyes*. Flow rates of the BGE and the sample solution were in the range of 3–4.5 mL min^{-1} and 2–5 µL min^{-1}, respectively. All sample inlets were closed except for the one used in the respective study. Sample outlets were only opened for the flow-profile-measurement but were closed otherwise (see **Figure S4**).

Measurement of a mixture of all three tracers (directory: evalmix/mix1)

Buffer flow rate and sample flow rate were 4.5 mL min^{-1} and 2 µL min^{-1}, respectively. 37 s after start of measurement the voltage was turned on (275 V, *i.e.* 56.1 V cm^{-1}; top: anode; bottom: cathode). Current measured was between 8 and 9 mA. 205 s after start of measurement the voltage was turned off. Only the centre inlet was open and all outlets were closed.

The corresponding video file reveals that the separation was successful but not stable (can be seen at about 200s). The reason was the huge bubble formation at both electrodes. Also, the stream does not go back to its neutral position but rather stays shifted even after the voltage is turned off (from 300s).

Measurement of a mixture of fluorescein and rhodamine (directory: evalmix/mix2)

Buffer flow rate and sample flow rate were 4.5 mL min^{-1} and 5 µL min^{-1}, respectively. Initially, there was no sample flow. 31 s after start of measurement the sample flow was started. At 631 s the voltage was turned on (150 V, *i.e.* 30.6 V cm^{-1}; top: anode; bottom: cathode). Current measured was stable at 4.3 mA. 826 s after start of measurement the voltage was turned off. During the measurement, the excitation light was slightly reduced at 768 s. Only the centre inlet was open and all outlets were closed.

The corresponding video file reveals that the sample inflow started at about 360 s. Also, the electric field was lower than in the previous experiment, there was still no stable separation (again due to bubble formation at both electrodes).

Measurement of flow profile with fluorescein only (directory: evalprofile)

Buffer flow rate and sample flow rate were 3.0 mL min^{-1} and 2 µL min^{-1}, respectively. The sample was introduced into one inlet (channel 1−5, numbered from bottom to top). All outlets were open. Initially, there was no sample flow. Sample flow was started shortly after each measurement started (at 71, 51, 21, 20, and 10 s for channel 1, 2, 3, 4, and 5, respectively) and stopped after the stream reached the right border of the separation zone (at 300, 327, 241, 353, and 142 s for channel 1, 2, 3, 4, and 5, respectively). No electric field was applied at any of these experiments.

Data format

A PNG file itself consists of several chunks represented by a four-letter code (*e.g.* IHDR, IDAT, tEXt). The code labels are case sensitive. The case of the first letter, for instance, marks the corresponding chunk as *critical* or *ancillary* chunks (for more information see the standard[3, 4]). *Critical* chunks are mandatory and hold the information for rendering an image. *Ancillary* chunks can hold any additional information, which is not critical for rendering the image – if a program encounters an ancillary chunk it does not understand it can safely ignore it. This allows the addition of metadata to a PNG file without breaking it.

We decided to use a twin-track approach in order to achieve both reliability and readability. First, a new, private chunk labelled **dfFe** (**d**ata of **f**ree **f**low **e**lectrophoresis) was introduced for reliably storing data. Secondly, the data stored in this chunk is mapped to the public tEXt-chunks for easy readability. Chunks are generated by serialization of a *Python* dictionary (an associative array), which contains information generated and put together during the various stages of experimentation and evaluation by the corresponding programs.

Table S1 gives a short overview of data the dfFe-chunk can hold and by which program this data is generated. The format is open, so it can easily be extended. For convenience, we developed a command-line tool **pngdata.py**, which can read and modify the **dfFe**-chunk as well as display all chunks present in any PNG file. This is used by the provided batch files to add the information about chip features into the data files. In order to better distinguish regular PNG files from ones with FFE data in them we use a double extension *.FFE.PNG* (instead of pure *.PNG*) for the latter. This ensures the compatibility with viewers (e.g. the preview function of Windows Explorer) and editors.

Table S1. Overview of properties in a dfFe-chunk. Please see the respective programs for information on the specific format of each property.

Property	**Description**	**generated by program**
Channels recorded	Channels recorded during acquisition	**acquire.py** (set in **setupcamera.py**)
Dataline 1	Data line 1 provided by user	**acquire.py** (set in **setupmeasurement.py**)
Dataline 2	Data line 2 provided by user	**acquire.py** (set in **setupmeasurement.py**)
Experiment name	Name of experiment provided by user	**acquire.py** (set in **setupmeasurement.py**)
Image counter	Number of image in set	**acquire.py**
Inlets	List of inlet coordinates and widths (in mm)	**manual (pngdata.py)**
Inner separation zone	Four coordinates in pixels of the separation zone (inner box of markings)	**findfeatures.py**
Outer separation zone	Four coordinates in pixels of the outer box around the markings	**findfeatures.py**
Outlets	List of outlet coordinates and widths (in mm)	**manual (pngdata.py)**
Physical zone dimensions	Physical zone dimensions (in mm)	**manual (pngdata.py)**
Snapshot time	Time in seconds elapsed since start of measurement	**acquire.py**
Timestamp	Timestamp of image acquisition	**acquire.py**
Total frame counter	Number of frames recorded and integrated	**acquire.py**
Trajectories	List of lists containing the trajectory points and widths (in pixels)	**findtrajectories.py**

Short description of Python programs

General

The software suite includes a function library (**ffe.py**) and some ready-to-use programs for setup, acquisition, processing, and evaluation of FFE experiments. The programs were written in *Python 2.7* and according to *Eric Raymond's 17 Unix Rules* (in particular following the *Rules of Simplicity and Modularity*).[5] Main modules used are *Numpy*, *Mathplotlib*, *Scipy*, and *OpenCV* (Open Source Computer Vision Library). The software suite is compatible with every camera supported by *OpenCV* itself including a huge variety of web-cameras (see documentation of *OpenCV*).

Most of the programs are console-based programs that can be controlled through command line arguments. For a list of arguments call the respective program with `--help`. All console-based programs were designed to allow the seamless integration into batch processes. Examples of these batch processes are included in the software package, *e.g.* for evaluating the flow profile of the FFE chip. The programs for setup the camera and measurement provide a *Graphical User Interface* (GUI).

Please note that the general descriptions below are meant for providing a rough overview. Please see the source code and the library documentation (**ffe.m.html**) for more details. The step numbers given below correspond to the numbers given in comments in the respective programs.

Acquire.py (console)

This programs records snapshots and a video using the camera (**setupcamera.py**) and measurement settings (**setupmeasurement.py**). It is automated and only provides a live view with almost no user interaction. Briefly, the program

0. setups the logging
1. opens and setup the camera device
2. reads and setup measurement parameters (including creating a directory for saving the data)
3. records and saves a background image
4. setups the video recording device
5. starts recording (integrating frames, creating overlay for live view and video, saving frames to images and video)
6. cleans up

Alignchip.py (GUI)

This program aids the user in performing the alignment of the chip (see also main text). Direct, visual feedback on the state of alignment is provided. Briefly, the program detects the separation zone and bounds it within a rectangle. It also draws a reference rectangle which represents the target (optimal) orientation of the separation zone in the imaging field of view. The overlap between the areas of the two rectangles is calculated, and presented as a percentage. If the alignment is perfect, both rectangles are equal and the overlap is 100%. The rectangles are rendered on a black canvas with using red (detected separation zone of the chip) and green (target orientation) filling colour, respectively. In the overlapping area the resulting colour is yellow. The overlap percentage is calculated by counting the yellow pixels and dividing it by the pixels of one of the rectangles. **Figure 2** in the main text shows a scheme for alignment.

Combineandrenderflow.py (console)

This program is a part of a quick&dirty example on developing advanced evaluation methods (here: flow profile) based on the data provided by trajectory-finding and separation-zone-detection. The program itself depends on the output of **timeposition.py**, which extracts the time-dependent position of the stream-fronts for each inlet-channel (1–5, see experimental details). The script reads the data from **flowch1-5.csv** (experimental input) as well as from **calculatedvelocityfield.txt** (COMSOL input) calculates the velocities, and plots flow profiles for both input sets (see **Figure 7** in main text).

Ffe.py (library)

This module is the core and heart of the presented software suite. It defines commonly used functions and algorithms for the image processing and analysis system and its programs. Please see the module documentation (**ffe.m.html**, which was generated by **pdoc**) and the source code for details.

Findfeatures.py (console)

This program opens a FFE.png file and tries to detect the separation zone markings as well as the flow markers. It will then save this information to the same PNG file. Briefly, the program

0. parses command-line arguments
1. setups the logging
2. extracts all contours from the image
3. tries to find the separation zone (see main text for a description)
4. tries to find the flow markers (see main text for a description)
5. saves the position of the separation zone and flow markers in the data file
6. cleans up

Findtrajectories.py (console)

This program opens a FFE.png file and tries to detect all trajectories. It will then save this information to the same PNG file. Briefly, the program

0. parses command-line arguments
1. setups the logging
2. reads input image and data (*e.g.* separation zone coordinates, physical resolution, etc.)
3. determines the inverse flow direction (based on markers, if present)
4. calculates the resolution (in mm per pixel)
5. creates a list of starting points from the list of inlets
6. tries to find trajectories for each channel separately
7. saves trajectories positions and widths (pixel based) in the data file
8. cleans up

Pngdata.py (console)

This program allows viewing and editing the dfFe chunk in a PNG file on the command line. It can be used to add information about features to the file for evaluation such as it is done for the example batches. Please see help page (`--help`) and source code for more details on this program.

Rendertrajectories.py (console)

This program opens a FFE.png file and renders the trajectories in this file to a plot. If no output file is given, the plot is presented directly to the user. The origin is set to the inlet. Trajectories are rendered according to the output style provided.

Resolution.py (console)

This program opens a FFE.png file, and extracts and renders the resolution in this file to a plot. If no output file is given, the plot is presented directly to the user. It will consider and render every possible combination of trajectories present in the data chunk of the file.

Setup.py (console)

This small program registers the **ffe.py**-module with the local Python installation (using **disutils**). See library documentation for more details.

Setupcamera.py (GUI)

This program helps the user to set up the camera for the measurement. Settings are saved to a file, which is used by the recording and alignment programs.

Setupmeasurement.py (GUI)

This program helps the user to set up the various parameters for the measurement. Settings are saved to a file, which is used by the recording program.

Timeposition.py (console)

This program is a part of a quick&dirty example on developing advanced evaluation methods (here: flow profile) based on the data provided by trajectory-finding and separation-zone-detection. This program extracts the time-dependent position of the stream front for each measured inlet-channel (1−5, see experimental details). The data is subsequently used by **combineandrenderflow.py** to render the flow profile of the chip.

References

1. M. J. Weber, *Handbook of Optical Materials*, Taylor & Francis, 2002.
2. F. J. Agostino, C. J. Evenhuis and S. N. Krylov, *J. Sep. Sci.*, 2011, **34**, 556−564.
3. T. Boutell, *RFC 2083*, 1997, DOI: 10.17487/rfc2083.
4. ISO, *Computer graphics and image processing - Portable Network Graphics (PNG)*, 2004, **15948**.
5. E. S. Raymond, *The Art of UNIX Programming*, Pearson Education, 2003.