Computer Physics Communications 265 (2021) 107987

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

Cerman ☆: Software for simulating streamer propagation in dielectric liquids based on the Townsend–Meek criterion ☆☆

I. Madshaven [a], O.L. Hestad [b], P.-O. Åstrand [a,*]

[a] Department of Chemistry, NTNU – Norwegian University of Science and Technology, 7491 Trondheim, Norway
[b] SINTEF Energy Research, 7465 Trondheim, Norway

ARTICLE INFO

Article history:
Received 22 July 2020
Received in revised form 8 January 2021
Accepted 30 March 2021
Available online 20 April 2021

Keywords:
Streamer breakdown
Dielectric liquid
Simulation model
Python
Computational physics

ABSTRACT

We present a software to simulate the propagation of positive streamers in dielectric liquids. Such liquids are commonly used for electric insulation of high-power equipment. We simulate electrical breakdown in a needle–plane geometry, where the needle and the extremities of the streamer are modeled by hyperboloids, which are used to calculate the electric field in the liquid. If the field is sufficiently high, electrons released from anions in the liquid can turn into electron avalanches, and the streamer propagates if an avalanche meets the Townsend–Meek criterion. The software is written entirely in Python and released under an MIT license. We also present a set of model simulations demonstrating the capability and versatility of the software.

1. Introduction

1.1. Streamers in liquids

Dielectric liquids, specifically transformer oils, are used as electric insulation in high-power equipment such as power transformers [1]. Equipment failure is always a possibility, and in a world with ever-growing need for energy, there is a continuous effort to make equipment better, cheaper, more compact, and more environmentally friendly. To prevent equipment failure due to electrical discharges, new insulating liquids as well as additives are tested, experiments are carried out to better understand the physical nature of the phenomena, and simulations are performed to test the validity of predictive models [2,3].

Since electrical discharge events are rare at operating conditions, model experiments are designed to induce discharge in the liquid. In one such model experiment, a needle electrode is placed opposing a planar electrode, where the needle–plane gap is insulated by a liquid [2]. If high voltage is applied, resulting in a sufficiently strong electric field close to the needle, the liquid will lose its insulating properties and begin to conduct electricity, and subsequent (partial) discharges from the needle into the liquid can occur. The charge transported into the liquid can increase the electric field and lead to partial discharges in new regions in a self-induced manner. Shadowgraphic images reveal that a gaseous channel, a "streamer", is formed and how it branches as it propagates through the liquid [4]. If a streamer bridges the gap between two electrodes, an electric discharge can follow, possibly destroying the affected equipment.

Streamers are commonly classified by their polarity and speed of propagation from the slow 1st mode to the fast 4th mode, ranging from below 0.1 km/s to well above 100 km/s [2]. Streamers with negative polarity typically have a lower inception voltage than streamers with positive polarity (positive streamers), however, positive streamers typically lead to breakdowns at lower voltage than negative streamers, and as such, research is mainly concerned with positive streamers. The streamer phenomenon involves processes covering several length and time scales. Speed and branching is studied in gaps of different sizes (mm–m), while many of the interesting processes, such as field ionization, high-field conduction, electro-hydrodynamic movement, bubble nucleation, cavitation, electron avalanches, photoionization, occur on a µm-scale [2,3]. A streamer usually stops or leads to a breakdown on a µs-scale (km/s = mm/µs), whereas processes such as recombination of electrons and anions can occur within picoseconds. Consequently, experimentation as well as simulation is challenging.

1.2. Modeling and simulations

While sophisticated equipment is required for experiments, simulations often investigate the effect of given processes through

☆ Available at: `github.com/madshaven/cerman`.
☆☆ The review of this paper was arranged by Prof. David W. Walker.
* Corresponding author.
E-mail address: per-olof.aastrand@ntnu.no (P.-O. Åstrand).

https://doi.org/10.1016/j.cpc.2021.107987

relatively simple models. The fractal nature of the streamer structure can be simulated by considering a lattice where each point is either part of an electrode, the liquid, or the streamer [5]. Here, the streamer expands to new lattice points when some criterion, such as electric field strength, is obtained. Through kinetic Monte Carlo methods, the stochastic nature and the physical time can also be studied in such simulations [6–8]. Radial expansion of streamer channels and conduction within the streamer have also been investigated [9–12]. The latter mechanism, channel conduction, is also included in [13,14], where the streamer propagation criterion is a function of the square of the electric field. The charge and conduction of a streamer can also be studied by considering the streamer as an electrical network of resistors and capacitors, without necessarily confining the points of the network to a grid [15]. Models where the streamer consists of a set of discrete points are simplistic but also efficient. Conversely, with a higher demand for computational power, computational fluid dynamic (CFD) methods can be applied to solve the equations for generation and transport of charged particles (the flow of natural particles is often ignored) during a streamer discharge [16,17], while the stochasticity and branching of streamers can be introduced by adding impurities [18]. Such CFD-calculations often ignore the phase change from liquid to gas as well and are confined to a small region because of the computational complexity. However, for simplified, single-channel streamers, both the phase change and the processes in the channel can be simulated [19,20]. Code used for simulation of streamers in liquids is rarely published, in fact, we found just a single example [21].

1.3. Avalanche model

We have previously described our streamer model for positive streamers where the propagation is based on an electron avalanche mechanism [22]. Here, the avalanche process takes place in the liquid, in the high-field region in front of the streamer, and can cause a direct transition to a streamer channel. We assume this is the main propagation mechanism of positive second-mode streamers in non-polar liquids. However, as mentioned above, there are other possible propagation mechanisms. The focus of the model is on the streamer extremities and the high-field region in the liquid in front of them, whereas the streamer channel itself, and the mechanisms therein, are given a simplified representation. The model has been extended to account for conductance in the streamer channel and capacitance between the streamer and the planar electrode [23], as well as photoionization in front of the streamer [24], the latter as a mechanism for the transition from slow to fast propagation.

Streamer propagation is simulated in a setup resembling model experiments, a needle–plane gap filled with a model liquid, see Fig. 1 for details. The needle and streamer give rise to an electric field, affecting charged particles in the liquid. A number of anions, "seeds" for electron avalanches, is modeled within a volume surrounding the streamer, a "region of interest" (ROI). Electrons released from the anions can create electron avalanches, and the streamer propagates when an avalanche meets the Townsend–Meek criterion, i.e. exceeds a critical number of electrons [22]. The needle and the streamer heads (the extremities of the streamer) are modeled as hyperboloids, which simplifies calculating the electrical field since the Laplacian is analytic in prolate spheroid coordinates [25]. The electric field and potential are calculated by considering electrostatic shielding [22], as well as the conductance in the channel and the capacitance towards the planar electrode [23]. The streamer undergoes a transition into a fast propagation mode when radiation from the streamer head can ionize molecules directly in front of the streamer [24]. More details on the model are given in section 3.

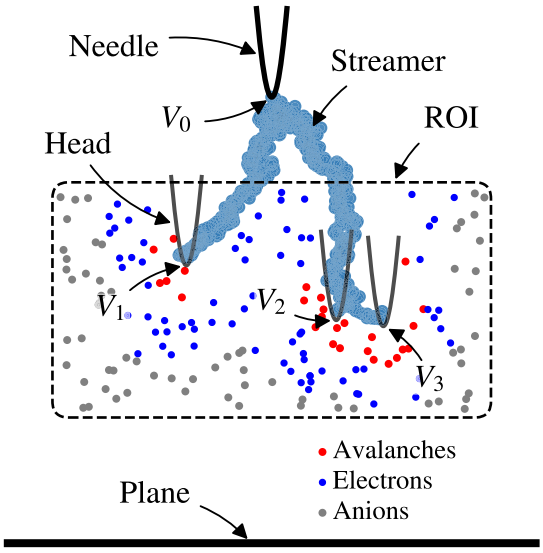

Fig. 1. Illustration of the main components in the simulation model. The needle electrode and the streamer heads are hyperboloids, each with a potential V_i. A region of interest (ROI) is used to limit the computational effort to a region surrounding the active part of streamer. The ROI controls the position of the "seeds", which are classified as anions, electrons, or avalanches, depending on the electric field strength at their position. The "shadowgraphic" image of the streamer is created by plotting all former positions of streamer heads. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

The main output of the simulations includes the propagation speed, the streamer shape (branching), and propagation distance. In addition, properties such as the initiation time, the potential of individual streamer heads, electric breakdown within the streamer channel, and avalanche growth, can also be investigated. Simulations show how various parameters affect the results, where the gap size, applied voltage, and type of liquid are important parameters for a simulation. Furthermore, other parameters such as the size of a streamer head, the conductivity of the streamer channel, properties of additives, and avalanche growth parameters can be varied to validate whether the underlying physical models are reasonable.

1.4. Scope

The present work describes the use, functionality and implementation of *Cerman* [26], a software to do simulations with our model [22–24], with the purpose to make the software publicly available. Section 2 demonstrates how to set up, simulate, and evaluate results of a relevant problem. Further details on the model and its implementation are given in section 3, whereas section 4 outlines the current functionality and some prospects for the future. A summary is then given in section 5. Furthermore, details on the algorithm are included in Appendix A, simulation parameters in Appendix B, and simulation example input files in Appendix C.

2. Simulation – using the software

2.1. Software overview

The software name *Cerman* is an abbreviation of *ceraunomancy*, which means to control lightning or to use lightning to gain information. The implementation is done in Python, an open-source, interpreted, high-level, dynamic programming language [27], and the software is available on GitHub [26] under an MIT license. The software is script-based, and controlled through the command `cerman`, which is used for creation of input files, running simulations, and evaluating the results. When a simulation is started,

Listing 1: Example of JSON-input file, `cmsim.json`, defining a simulation series with several values for the needle voltage V_0 and the threshold for breakdown in the streamer channel E_{bd}, both with and without photoionization enabled. Furthermore, each permutation is to be carried out 10 times with different initial seed positions. Note, by setting `alphakind` to `A1991`, α is calculated by (6). See Appendix B for a description of the parameters.

```
{
    "gap_size": 0.010,
    "needle_radius": 6e-06,
    "needle_voltage": "linspace(60e3, 150e3, 10)",
    "liquid_name": "cyclohexane",
    "alphakind": "A1991",
    "liquid_Ealpha": 0.65e09,
    "liquid_IP": 9,
    "streamer_head_radius": 3e-06,
    "streamer_U_grad": 0e+06,
    "streamer_d_merge": 12.5e-6,
    "streamer_scale_tvl": 0.1,
    "streamer_photo_enabled": [false, true],
    "streamer_photo_efield": 3.1e9,
    "streamer_photo_speed": 20e3,
    "rc_tau0": 1e-6,
    "rc_breakdown": [0e6, 4e6, 8e6, 16e6, 64e6],
    "random_seed": 1,
    "simulation_runs": 10,
    "save_specs_enabled": {
        "stat": true,
        "gp5": true
    }
}
```

the simulation parameters are loaded from an input file, and the classes for the various functions are initiated. See Appendix B for a summary of simulation parameters. The simulation itself is essentially a loop where seeds in the liquid are moved and the streamer structure is updated until the streamer stops or leads to a breakdown. The algorithm is detailed in Appendix A.

2.2. Getting started

Download Cerman from GitHub [26] and install it by running

```
pip install .
```

from the downloaded folder. This installs the python package and the script `cerman`. Python 3.6 or above is required, as well as the packages `numpy` [28], `scipy` [29], `matplotlib`, `simplejson`, and `statsmodels`. The dependencies are automatically installed by `pip`. The software has been developed in OSX and has been tested on Linux as well.

2.3. Create simulation input

Each simulation requires a JSON-formatted input file where the parameters are given. Such files can be created from a master input file, specifying the parameters for a simulation series. A master input file can be created from a regular input file by changing a parameter value into a list of values. More than one parameter can contain lists, and all possible combinations of values are found to create the input files for the simulation series. To demonstrate, we use `cmsim.json` in Listing 1, which specifies a simulation series exploring the influence of various parameters. The ten values for the applied voltage are specified through a `linspace`-command, while the values for the threshold for breakdown in the channel and photoionization (fast mode) enabled are given in list form. Furthermore, `simulation_runs` specifies the number of similar simulations, only differing by `random_seed`. If `random_seed` is `null`, then each input file is created with a random number as `random_seed`, and when a number is specified, a range of number is generated, in this case, the numbers 1 through 10. Note that `random_seed` refers to the seed number for initializing the random number generator, not to the seeds within the ROI. However, a given `random_seed` does corresponds to a given initial positions of the seed anions. Defining fixed a `random_seed` for each simulation series makes it easier to see how a change in a given value affects the simulation, but for analyzing a larger assemble of simulations, it is usually preferable that the simulations are uncorrelated, i.e. have random initial anion placement.

Individual input files are created by running the `cerman` with the argument `ci` (create input) and specifying which file to expand with `-f`:

```
cerman ci -f <filename>
```

This command creates a number of new files by permutation of all lists in the given input file. The permutation of 10 random seeds, 10 applied voltages, 5 breakdown thresholds, and 2 modes for photoionization in Listing 1 results in 1000 files. By default, the random seed is expanded first, followed by the needle voltage, which is useful to consider when designing a simulation series. Choosing an appropriate number of values for these two parameters makes it easier to search for simulation files with given properties. When expanding the example in Listing 1, the least significant digit `??X` indicates random seed number, the second digit `?X?` indicates the needle voltage, while the most significant digit `X??` indicates the threshold for breakdown in the streamer channel and whether photoionization is enabled or not.

The action `pp` (plot parameters) creates a matrix representation of the parameter variation in set of input files, and is used like

```
cerman pp -g <pattern>
```

The argument `-g` specifies the pattern to search (or "glob") for, e.g. `cmsim_?00.json`. The files are plotted at the x-axis and the

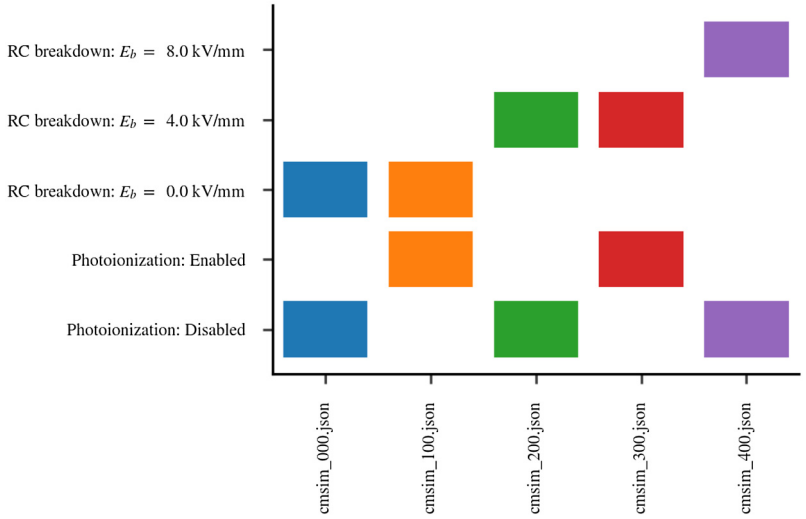

Fig. 2. Visualization of the difference in parameter values between a selection of input files.

varied parameters on the y-axis, see Fig. 2 for an example output. The name of the output file is based on the first file in the pattern.

After ensuring that the input parameter values are as desired, simulations are run using

```
cerman sims -g <pattern> -m <no>
```

which creates a queue of all files matching the pattern, and simulates a given number in parallel. For instance, `cerman sims -g "cmsim_?5?.json" -m 15`, simulates all input files with the same voltage, creating a queue where up to 15 separate subprocesses each run one simulation. These python processes are single threaded, and work best if `numpy` is limited to a single thread as well. Each simulation dumps the input parameters and progress information to a log file. For each simulation we then have a parameter file, a log file, and one or more save files. The files are named by extending the name of the master file, e.g.

```
cmsim.json              # master file
cmsim_290.json          # input parameters
cmsim_290.log           # log file
cmsim_290_gp5.pkl       # save file
cmsim_290_stat.pkl      # save file
```

2.4. Evaluate results

The results are evaluated by parsing the data stored from one or more simulations. The input file, Listing 1, defines two "save specifications" as `true`, i.e. enabled, each defining various simulation data to be dumped to disk at given intervals or occasions. The `save_spec` called `stat` saves iteration number, CPU time, simulation time, leading head z position, number of critical avalanches, and the position of each new streamer head, for every iteration. This enables evaluation of the shape of the streamer and its propagation speed. The `save_spec` called `gp5` saves most of the data available, including the position of all the seeds (anions/-electrons/avalanches), for every 5 percent of streamer propagation, i.e. a total of 20 times for a breakdown streamer. The data is saved using `pickle` and can be loaded to analyze a given iteration, a whole simulation, or by combining data from several simulations.

Evaluate iterations. Iteration data can be used to analyze the details of a simulation. This is particularly useful when evaluating the validity of the simulation parameters. Use for instance

```
cerman pi seedheads -r <start stop> -g <pattern>
```

where `pi` means "plot iteration" and `seedheads` is a scatter plot of avalanches and the streamer head configuration. The option `-r` controls the range of iterations to plot. Figure 28 in [22] shows a number of such plots.

Evaluate simulations. Use `ps` for simulation plots. These are mainly plotted with the z-position in the gap on the y-axis. Plotting the x- or y-position of streamer heads on the x-axis, using `shadow`, gives a "shadowgraphic" plot of the streamer:

```
cerman ps shadow -g <pattern> -o <options>
```

Similarly, plotting the propagation time on the x-axis is done in a `streak` plot (see Fig. 3). Options can be added to control the limits/extents of the plot, the figure size, the behavior of the legend, redefining the axis labels, starting each plot with an offset, saving the plotted data to a JSON-file, and much more. Use `help` to show available commands and options, for instance:

```
cerman help                 # for the main script
cerman ps help              # for plot simulation
cerman ps shadow -o help    # for shadow plot
```

Single simulation data may be of interest, but it is often better to compare several simulations in the same plot to visualize how the input parameters affect the results. The `gp5` save requires a lot of disk space, but can be very useful in analyzing the data. For plotting the potential of each head, use

```
cerman ps headsestr -g <pattern> -o <options>
```

The current (active) heads of the streamer are selected, their potential is scaled (electrostatic shielding, using a nnls-approach, cf. [22]), and then, the electric field at their position is calculated. However, the options can be used to specify the method for scaling and which positions to calculate the field for, e.g. at the position of each appended (new) streamer head. The electric field strength and electric potential are presented as scatter plots against the z-position of each given position, as well as dashed line indicating the average value, see Fig. 4.

Evaluate a series of simulations. The difference in the simulated parameters can be visualized using `pp` and globing for the `pkl`-files or `log`-files. An existing `log`-file indicates that a simulation was initiated. The command

```
cerman psr -g <pattern>
```

parses log files and plots the reasons why the simulations were terminated. An example of such a plot is shown in Fig. 5. This is a good way to verify that simulations have completed successfully.

When all simulations are completed and verified, parse all the save files and build a combined database of the results:

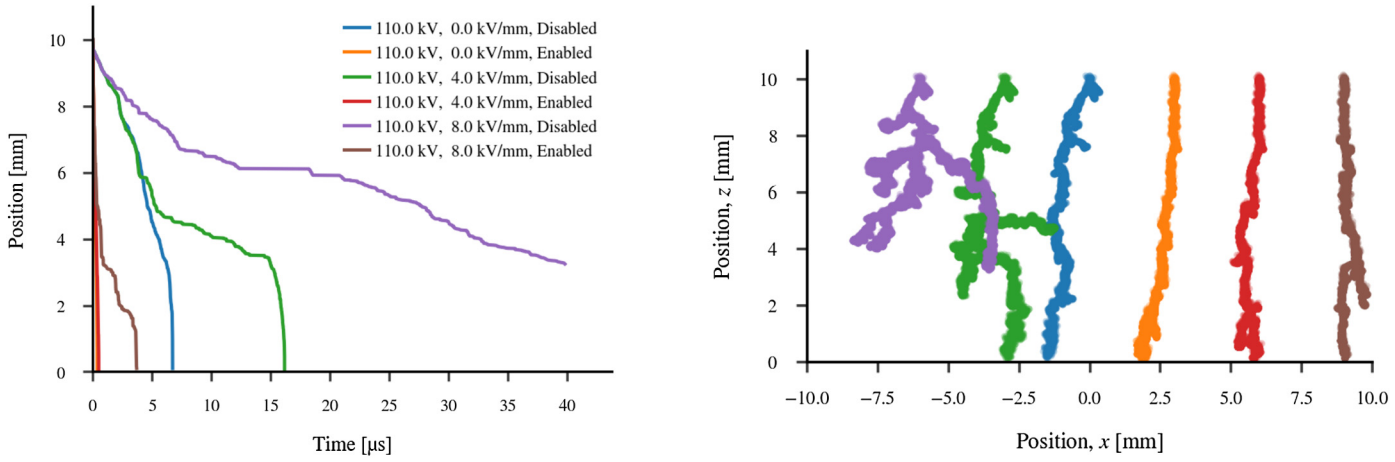

Fig. 3. (left) Streak plots where "options" have been used to limit the x-axis and to show V, E_{bd}, and photoionization on the legend. (right) Shadow plots where each streamer is plotted with an offset and the legend is hidden. The legend is the same for both plots.

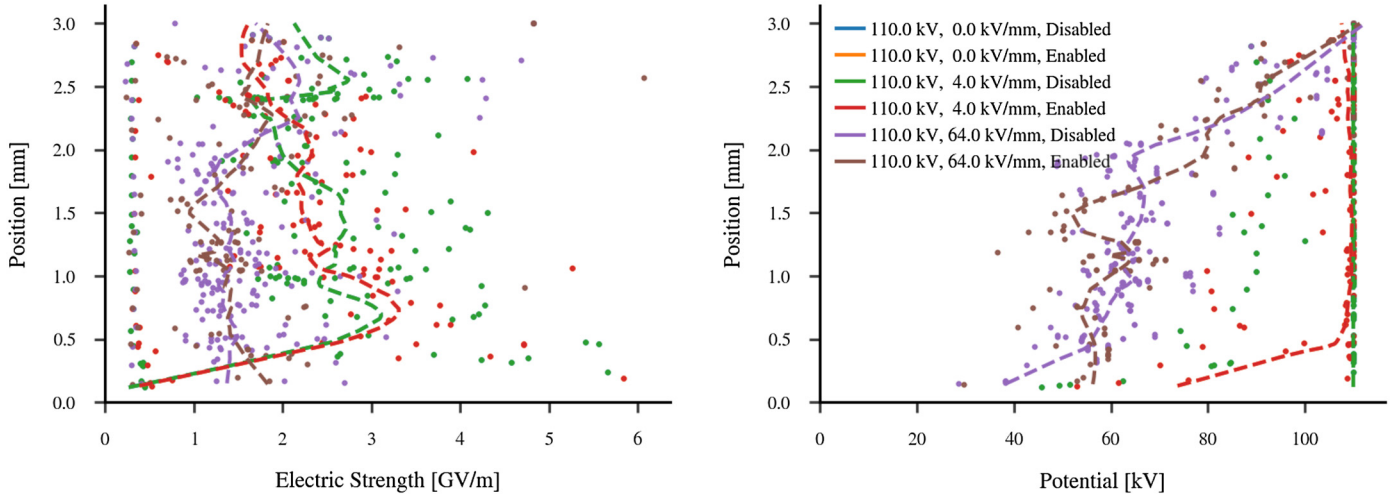

Fig. 4. The electric strength (left) and the electric potential (right) at the tip of each new streamer head. The streamer heads are sampled for every 5% of propagation. The "options" are used to control which streamer heads to use for the calculation and which positions to calculate for.

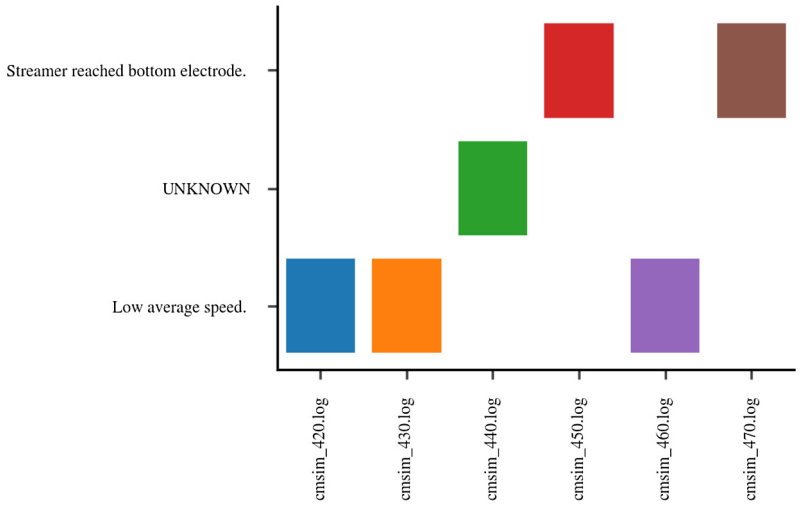

Fig. 5. Example, parsing the log files to visualize how simulations have terminated. "Unknown" implies that the simulation is not complete (ongoing or aborted).

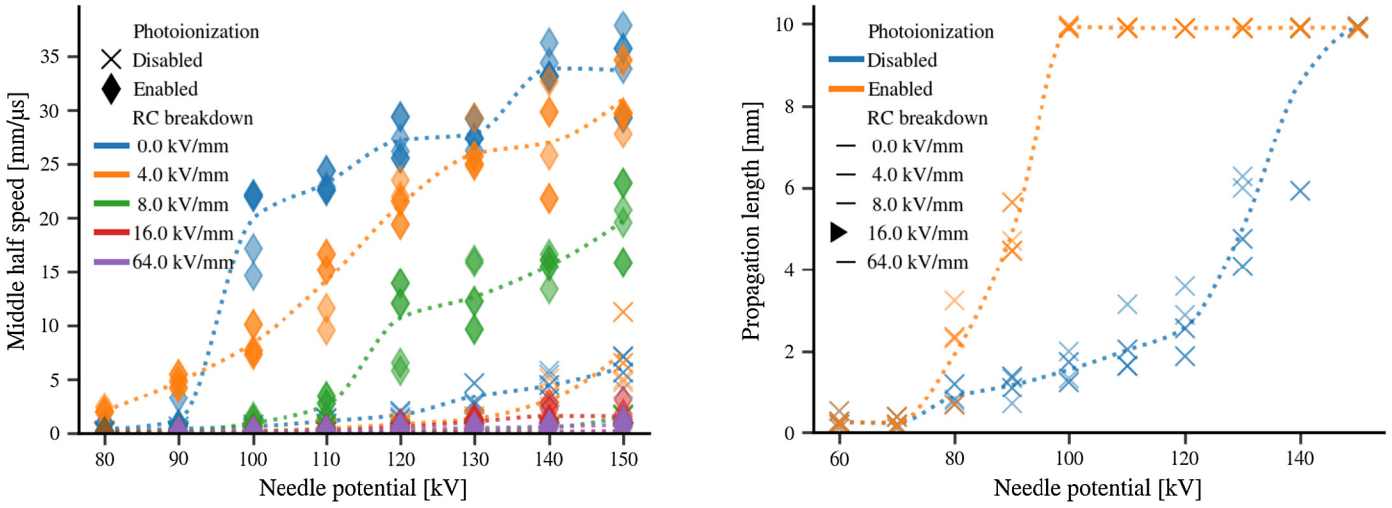

Fig. 6. Example, combination plots: (left) propagation speed using both markers and colors, and (right) propagation length for only a given value of one parameter with the other parameter colored.

```
cerman ca -g <pattern> -o mode=reload
```

The option `reload` forces files to be parsed, even if they already are in the database. When parsing the save files, a number of properties are extracted or calculated. Results such as propagation length (ls), average propagation speed (psa), inception time (ita), average jump distance (jda), simulation time (st), and computational time (ct) are saved in the database. The results are plotted by using

```
cerman pr <parameter> -f <file> -o <options>
```

The parameter gives the x-axis of the plot, for instance `v` (needle voltage). The results to plot is given through the option, e.g. `-o ykeys=ls_psa`. The software inspects the database of parsed save files for varied parameters and automatically create plots of all possible permutations using colors and markers. Fig. 6 shows a selection of such plots, where needle voltage is on the x-axis and the other parameters have been added automatically.

As illustrated, the software has been designed to facilitate simulating and analyzing a large simulations series in a semi-automatic fashion. The individual simulations have their parameters defined in dedicated files, while the behavior of the `cerman`-script is defined by command line arguments.

3. Model implementation

This section describes the implementation of our model [22–24] in more detail. An overview of the model is already given in section 1.3, and Fig. 1 is useful for understanding the principal setup, a needle–plane gap where electron avalanches grow from single electron seeds.

The electric field. The needle electrode and the streamer heads are modeled as hyperboloids. The Laplacian electric potential V_i and electric field $\boldsymbol{E}_i$ from a streamer head i is calculated analytically in prolate spheroid coordinates [22]. For a position $\boldsymbol{r}$, the electric potential $V(\boldsymbol{r})$ and field $\boldsymbol{E}(\boldsymbol{r})$ is given by the superposition principle,

$$V(\boldsymbol{r}) = \sum_i k_i V_i(\boldsymbol{r}) \quad \text{and} \quad \boldsymbol{E}(\boldsymbol{r}) = \sum_i k_i \boldsymbol{E}_i(\boldsymbol{r}), \tag{1}$$

where the coefficients k_i are introduced to account for electrostatic shielding. An optimization is performed such that the unshielded potential at the tip of any streamer head j equals the sum of all the shielded potentials at that position, $V_j(\boldsymbol{r}_j) = \sum k_i V_i(\boldsymbol{r}_j)$ [22]. The needle and the extremities of the streamer, i.e. each electrical hyperboloid, probably interacts to a greater extent than the method of superimposing and scaling the hyperboloids accounts for. To include such interactions or to calculate the full Poisson field requires a much greater computational effort. However, the applied method scales the potential at the very tip of each hyperboloid, to better estimate the Laplacian field in its vicinity.

Region of interest (ROI). The ROI is a cylindrical volume used to control the position of seeds, see Fig. 1. Note, the *seeds* in this respect include all anions, electrons, and even all avalanches. The number of seeds in a simulation is given by the specified density of seeds and the volume of the ROI. Initially, the seeds are placed at random positions within the ROI. When a seed falls behind the ROI, collides with the streamer, or creates a critical avalanche, it is removed and replaced by a new seed. The new seed is placed with a z-position one ROI-height closer to the plane, at a random xy-position (at a radius less than the ROI radius). The ROI volume is defined by a distance from the z-axis, and a given length above and below the leading streamer head, which is the part of the streamer closest to the planar electrode. When a new streamer head is created closer to the plane, the streamer propagates, and the ROI moves as well.

Anions, electrons, and electron avalanches. Each seed j is classified according to the electric field E_j at its location:

$$E_j \geq E_\mathrm{c} \implies \text{avalanche}, \quad E_j \geq E_\mathrm{d} \implies \text{electron}, \quad \text{or}$$
$$E_j < E_\mathrm{d} \implies \text{anion}, \tag{2}$$

where the avalanche threshold E_c and detachment threshold E_d are given as input parameters. Seeds move in the electric field with a speed dependent on their mobility μ, which gives the distance the seed moves, $\Delta \boldsymbol{s} = \boldsymbol{E}\mu\Delta t$, for a time step Δt. The number of electrons N_e in an electron avalanche, starting from a single electron, can be calculated by [22]

$$N_\mathrm{e} = \exp \sum_i \Delta s_i \alpha_i = \exp \sum_i E_i \, \mu_\mathrm{e} \, \alpha_\mathrm{m} \, e^{-E_\alpha / E_i} \Delta t_i, \tag{3}$$

where α is the avalanche growth, μ_e is the electron mobility, Δt is the time step, and i is an iteration, whereas the avalanche growth parameters α_m and E_α have been obtained from experiments [22]. In practice, however, we only calculate and store the exponent in (3), $Q_\mathrm{e} = \ln N_\mathrm{e}$,

$$Q_\mathrm{e} = \sum_i \Delta Q_i = \sum_i \Delta s_i \alpha_i, \tag{4}$$

for each electron avalanche. When a low-IP additive is present, α is modified by adding a factor [22]

$$\alpha_{i,\mathrm{add}} = \alpha_i(1 - x_{\mathrm{add}} + x_{\mathrm{add}} e^{k_\alpha (I_\mathrm{b} - I_\mathrm{a})}) \tag{5}$$

which is dependent on the mole fraction of the additive x_{add}, and the difference in IP between the base liquid I_b and the additive I_a modified by a factor k_α as prescribed by [30]. This is the default setting for the software. Another model for α given in [31] is also implemented:

$$\alpha_{i,\mathrm{mod}} = \frac{3eE_\alpha^2}{I_\mathrm{b} E_i} e^{-\frac{E_\alpha}{E_i}} . \tag{6}$$

This method is applied in Listing 1.

Expanding the streamer structure. According to the Townsend–Meek criterion [32], streamer breakdown occurs when an avalanche exceeds a critical number of electrons $N_\mathrm{c} = \exp(Q_\mathrm{c})$. When an avalanche obtains $Q_\mathrm{e} > Q_\mathrm{c}$, we place a new streamer head at its position [22]. The initial potential of a new streamer head is calculated by considering the capacitance and potential of the closest existing streamer heads [23]. If adding the new head implies removing another head (see the paragraph below), the potential changes slightly, mimicking transfer of charge. However, if both the new and the present head stays, they share the "charge", which gives a moderate reduction in the potential of both heads.

Optimizing the streamer structure. There are three criteria for removing heads. A streamer head i is removed if

$$\begin{aligned} &\nu_j(\boldsymbol{r}_i) < \nu_j(\boldsymbol{r}_j) \quad \text{or} \\ &k_i < k_\mathrm{c} \quad \text{or} \quad \big((|\boldsymbol{r}_i - \boldsymbol{r}_j| < d_\mathrm{m}) \text{ and } (z_i > z_j)\big) , \end{aligned} \tag{7}$$

are satisfied for any other streamer head j [22]. The first condition checks whether the tip of one hyperbole ($\boldsymbol{r}_i$) is inside another hyperbole, a collision (the ν-coordinate describes a hyperboloid, specifically the asymptotic angle). The second condition removes heads whose potential are to a high degree shielded by other heads (if the coefficient k_i in (1) lower than a threshold k_c). The third condition checks whether two streamer heads are closer than d_m and should be merged to a single head, where the one at the highest z-coordinate is removed.

Conduction and breakdown in the streamer channel. Conduction in the streamer channel increases the potential of each streamer head i each iteration,

$$\Delta V_i = V_0 - V_i \quad \rightarrow \quad V_i = V_0 - \Delta V_i \, e^{-\Delta t/\tau_i} . \tag{8}$$

The time constant $\tau_i = RC\tau_0$ (for a given head i) is calculated from the resistance R in the channel and the capacitance C towards the plane [23]

$$R \propto \ell , \quad \text{and} \quad C \propto \left(\ln \frac{4z + 2r_\mathrm{s}}{r_\mathrm{s}} \right)^{-1} , \tag{9}$$

where ℓ is the channel length (distance to the needle), r_s is the tip curvature radius of the streamer head, and z is the z-position of the streamer head. As such, each streamer head is treated as an individual RC-circuit, e.g. the three streamer heads in Fig. 1 would each have an individual resistance (channel) as well as an individual capacitance towards the planar electrode. The capacitance in (9) is based on a single hyperboloid above a grounded plane [23]. As such, superimposing each streamer branch would overestimate the capacitance of a branched streamer, and therefore this calculation is not performed within our simulations. In fact, we do not even calculate the absolute capacitance of individual branches, the capacitance of a streamer head is always kept as a measure relative to either the needle electrode or another part of the streamer. In addition to the calculation of t_i in (8), the capacitance is used to determine the change in streamer head potential when the streamer propagates to new positions [23]. The linear resistance in (9) is a simple model in comparison to models which also estimates expansion and relaxation within the channels, e.g. [11,14]. However, we also model breakdowns within the streamer channel, for which the resistance is greatly reduced. If the conduction is low, the potential difference ΔV_i increases as the streamer propagates, and the electric field within the streamer channel $E_\mathrm{s} = \Delta V_i / \ell_i$ may increase as well. If E_s exceeds a threshold E_b, a breakdown occurs, equalizing the potential of the streamer head and the needle, which is achieved by setting $\tau_i = 0$ in (8) for the given iteration.

Photoionization. Photoionization is a possible mechanism for fast streamer propagation [33]. We have proposed a mechanism in which the propagation speed of a streamer increases if the liquid cannot absorb radiation energy to excited states, as a result of a strong electric field reducing the ionization potential [24]. Since the full model, considering fluorescent radiation from the streamer head, and a field-dependent photoionization absorption cross section, is computationally expensive, a simpler model is used in the simulations. Instead we calculate the field strength E_w required to reduce the ionization potential below the energy of the fluorescent radiation. In each iteration, if the electric field E at the tip of a streamer head i exceeds the parameter E_w, the head is moved a distance $\Delta \boldsymbol{s}_i$ towards the planar electrode,

$$\Delta \boldsymbol{s}_i = -v_w \, \Delta t \, \Theta(E(\boldsymbol{r}_i) - E_w) \, \hat{\boldsymbol{z}} , \tag{10}$$

where v_w is the photoionization speed, and Θ is the Heaviside step function. For more details on the entire model, see our previous work [22–24].

4. Current functionality and future prospects

The main function of the model and the software is to simulate streamers in a point–plane gap, using the Townsend–Meek criterion for propagation. The propagation criterion is met when electron avalanches obtain a given size. This model and the algorithm are fixed, but there are several parameters which can be adjusted. Changing experiment features such as needle tip radius, gap size, voltage, liquid properties, or the parameters of the algorithms, is straightforward. Proposals to extend the software to encompass new functionality is given in this section.

In [22] we explored the fundamental features of the model, i.e. a streamer consisting of charged hyperbolic streamer heads, and streamer growth by electron avalanches initiating from anions. The model predicts several aspects of streamer propagation, and shows how they are linked towards the values of given input parameters. The predicted propagation speed and the degree of branching were both lower than expected. We found how the speed was dependent mainly on the number of electron avalanches and their growth, while the branching was mainly related to how the streamer heads were configured and managed, which is mainly controlled by the parameters k_c and d_m in (7).

The electron avalanche model was calibrated to have streamer inception at 30 kV [22] according the propagation criterion given in [30]. However, the simulated streamers do not propagate far at this voltage, and potentials above 60 kV is required for a breakdown [22]. For small gaps, 3 mm to 10 mm, the inception of propagation streamers occurs between 14 kV to 36 kV, depending on how "propagation" is defined [30,34,35]. The higher end of these values can typically also give breakdown, whereas fast streamers can occur at voltages above 60 kV [35]. In larger gaps, 50 mm to 77 mm, well above 100 kV is required for a breakdown, and the acceleration voltage is just 10 kV to 30 kV above that

[34,36]. The focus of our work has been on using correct values for our parameters, rather than, for instance, argue for an increase in n_{seed} or μ_e to increase the speed of the simulated streamers.

When new streamer heads were added, their potential was set assuming a constant electric field within the channel, resulting in a moderate voltage drop between the needle electrode and the streamer head [22]. To better represent the dynamics of the streamer channel, an RC-model was developed [23]. In the RC-model, the potential of new streamer heads is dependent of the potential of the closest existing streamer heads. If the conductance of the streamer channel is high, then the potential of the streamer head is kept close to that of the needle, giving results comparable to those without the RC-model. Conversely, having low conduction regulates the speed of the streamer, increasing the likelihood that more branches are able to propagate. Furthermore, the RC-model also allows for simulation of a breakdown within the streamer channel itself, which is likely what occurs during a re-illumination. This breakdown occurs when the electric field within the channel exceeds a given threshold.

The importance of photoionization during a streamer breakdown is unknown. We explored different aspects of photoionization in [24], and implemented a model for change to a fast propagating mode. Molecules excited by energetic processes, such as electron avalanches, can relax to a lower energy state by emitting radiation. We argued that fluorescent radiation can be important, and modeled how this radiation can cause ionization in high-field areas, since the high-field reduces the ionization potential. For instance, for cyclohexane, we assume that radiation from the lowest electronically excited state (about 7 eV) can cause ionization when the local electric field is in the range 1.4 GV/m to 3.1 GV/m [24]. Molecular dissociation within the streamer and radiation from other molecules than the base molecules are interesting aspects of the streamer phenomenon, but are not considered in the current model. Moreover, the energy available in form of radiation is unknown and difficult to quantify.

In the current implementation, a square wave voltage is applied to the needle at the beginning of the simulation. It is easy to change the behavior to a voltage ramp, from zero to max over a given time. This can be the basis for a study on streamer inception where other parameters such as needle size and the electron properties of the liquid are investigated as well. Simulating a dynamic voltage, such as a lightning impulse, requires some more work, but is also achievable.

We have focused on cyclohexane since many of its properties are well-known, but other non-polar insulating liquids can be studied by changing relevant parameters. The seed density n_{ion} is based on the low-field conductivity σ and electron mobility μ_e of the liquid, and the propagation speed scales linearly with both seed density n_{ion} and electron mobility μ_e. In [22], we calculated n_{ion} in the range 6×10^{11} m^{-3} to 6×10^{12} m^{-3}, the lower value from the equilibrium generated by cosmic radiation, and the higher value from the ion mobility and the liquid conductivity. For simulations thereafter, $n_{\text{ion}} = 2 \times 10^{12}$ m^{-3} have been used. The electron avalanche growth parameters are also liquid-dependent, and E_α in particular has a big impact on the results [22]. Streamer parameters, such as conductivity of the streamer channel and streamer head radius, need to be reevaluated as well for other liquids. The properties of the streamer channel are also important to simulate the effects of external pressure, which mainly affects processes in the gaseous phase [37].

The effect of additives with a low ionization potential (IP) is modeled as causing an increase in electron avalanche growth [22,30]. Other additives can easily be used as long as the IP of both the base liquid and the additive are known. Low-IP additives are known to facilitate the propagation of slow streamers and to increase the acceleration voltage, possibly as a result of increased branching [34], however the mechanisms involved are not known. It is possible that low-IP additives are sources of electrons that can initiate avalanches, produced for instance through photoionization or fluctuations in the electric field. Such mechanisms can be added to the model and simulated, but will require some work. Furthermore, the mechanisms of added electron scavengers can also be interesting for further investigation, and particularly if negative streamers are to be simulated [30].

Our primary concern has been with positive streamers, since these are more likely to lead to a complete breakdown than negative streamers. The model relies on electrons detaching from anions, moving towards regions where the field is higher, and then forming electron avalanches. The polarity of our model can easily be reversed, however, the electrons would then drift away and be unable to form avalanches. As such, a model for creation of new electrons is needed to simulate negative streamers with the software. Charge injection from the needle and the streamer can be one such mechanism [38]. Another option is to model electron generation (charge separation) in the high-field region surrounding the needle and the streamer. Such mechanisms are interesting for simulating positive streamers as well.

The hyperbole approximation simplifies the calculations of the electric field, both from the needle electrode and from the streamer heads. Other experimental geometries, such as plane–needle–plane, or even more realistic real-life geometries can be implemented. The challenge is to set the correct shielding or scaling of the streamer heads according to the influence of the geometry. Simpler geometric restrictions are easier to implement, for instance by manipulating the ROI. The streamer can be restricted to a tube by setting a low value for the maximum radius of the ROI. Another method is setting the “merge threshold” very high, such that the streamer is restricted to a single channel with a single head, which can be representative for a streamer in a tube [39].

There are many mechanisms that can be added to investigate different methods of streamer propagation, for instance effects of Joule heating or electro-hydrodynamic cavitation [40]. There are also several parts of the existing model that can be improved. Better calculation and balancing of charges and energy would greatly improve the model. For instance an electric network model where the streamer is consisting of several interconnected parts, in contrast to the current implementation where all the streamer heads are individually “connected” to the needle. Such an approach can give a better understanding of the charge flow in the different parts and branches of the streamer, as well as a better representation of the electric field. Development towards a model for a space-charge limited field [41] can further improve the electric field representation, however, possibly at a high computational cost.

5. Summary

We present a software for simulating the propagation of positive streamers in a needle–plane gap insulated by a dielectric liquid. The model is based on the Townsend–Meek criterion in which an electron avalanche has to obtain a given size for the streamer to propagate. The software was developed and used for simulating our models for electron avalanche growth [22], conductance and capacitance in the streamer channels [23], as well as photoionization in front of the streamer [24]. From the examples on how to set up, run, and evaluate simulations, others can recreate our previous results or create their own set of simulations. Furthermore, the overview of the implementation and algorithm serves as a good starting point for others to change or extend the functionality of the software.

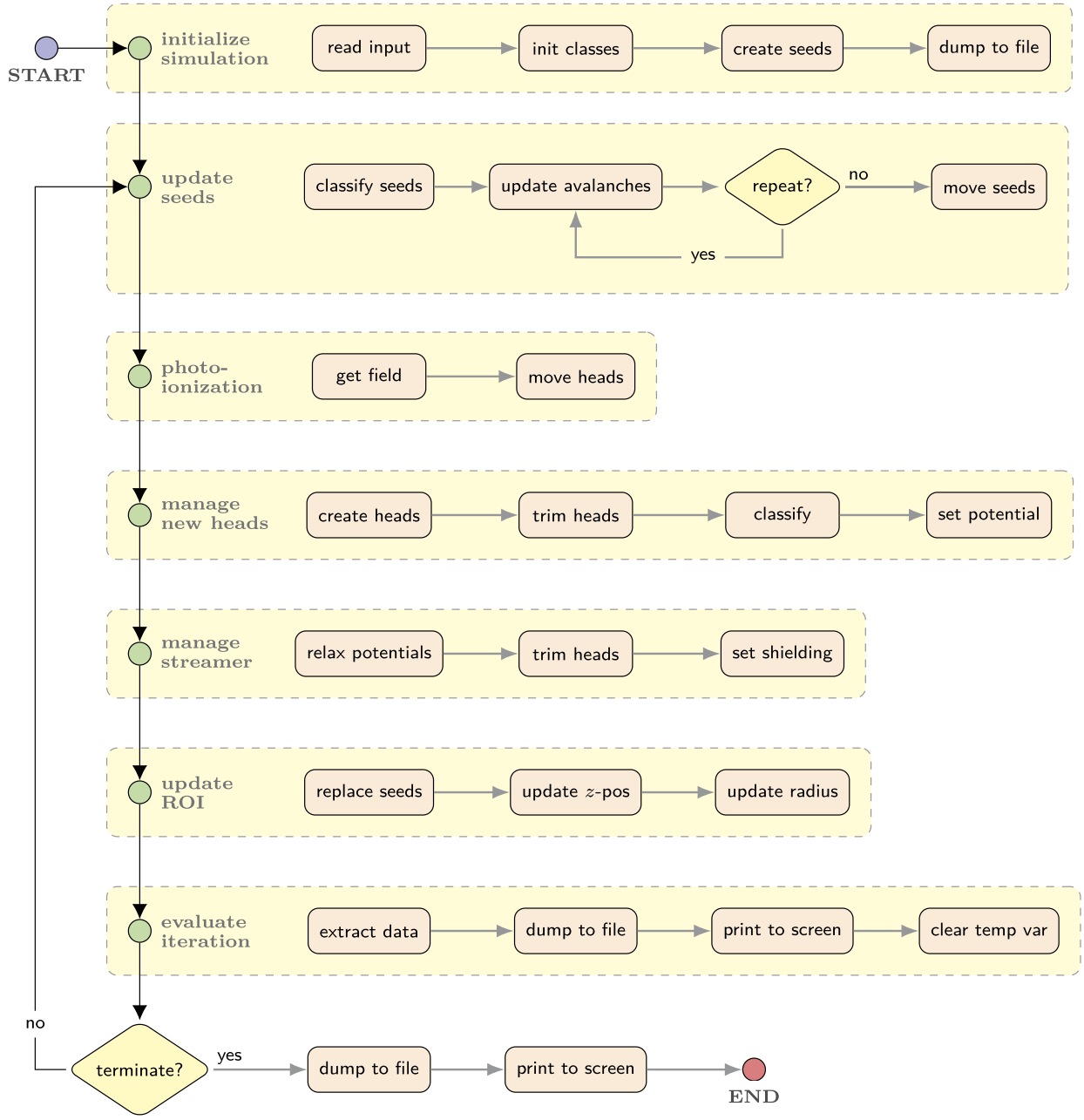

Fig. 7. The simulation algorithm consists of initialization, a loop where the seeds and the streamer structure is updated, and then a finalization. In the loop, first the seeds (anions, electrons avalanches) are affected by the electric field, then the streamer structure is modified and this changes the electric field, finally the region of interest (ROI) is updated and the data from the iteration is evaluated. The loop concludes when one (of several) criterion is met, typically low propagation speed or reaching the opposing electrode. Details on each step are found in Appendix A.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work has been supported by The Research Council of Norway (RCN), ABB and Statnett, under the RCN contract 228850.

Appendix A. The algorithm

This section describes the algorithm used to implement the model in more detail, essentially each part of Fig. 7, while referencing relevant parts from section 3.

Initialize simulation. The simulation input parameters are read and used to initialize classes for code profiling, simulation logging, calculation of avalanche growth, the needle, the streamer, the region of interest (ROI), the seeds (anions, electron, avalanches), how to save data, and how to evaluate simulation data. The initiation of the log file includes dumping the input parameters to the file. Given the ROI volume and the seed density, a number of seeds is created and placed within the ROI at random positions. Then, the save files are initialized by dumping the initial data, mainly information concerning the needle and the seeds.

Update seeds. The electric field E is calculated for each seed (applying (1)) and the seeds are classified as avalanches, electrons, and anions. All the avalanches move in the electric field and grow in size (see (3)). The procedure is repeated until an avalanche collides with a streamer head, an avalanche meets the Townsend–

Meek criterion, or a total of N_{MSN} repetitions has been performed. The "time" spent in this inner loop sets the time step for the current iteration of the main simulation loop. Finally, all the electrons and anions are moved. The inner loop over just the avalanches saves significant computational time since the calculation of the electric field is the most expensive part of the computation and the avalanches are usually a small fraction of all the seeds.

Photoionization. The electric field at the tip of each streamer head is calculated and compared with the threshold for photoionization. Each streamer head where $E > E_w$ is "moved" a distance $v_w \Delta t$ towards $z = 0$. Moving implies creating a new head, setting the potential by "transferring charge", and removing the old head.

Manage new heads. For each critical avalanche a new head is created. If the simulation time step is set sufficiently low, there is usually zero or just one new head. The new head is discarded if it satisfies any of the criteria in (7), however, if adding it will cause another to be removed (later), the new head is classified as "merging". If none of the criteria in (7) is met by adding the new head, i.e. all heads are kept, then it is a "branching" head, since adding it is potentially the start of a new branch. The potential of "merging" is set by "charge transfer" from the closest head, while "branching" heads have their potential set by "sharing charge" with the closest head, where the latter method also modifies the potential of the existing head [23].

Manage streamer. The potential of each streamer head is relaxed towards the potential of the needle by applying (8). This step increases the potential of the streamer head to the potential of the needle when a breakdown in the channel occurs. As mentioned above, the calculation of the electric field for the seeds is computationally expensive, and it actually scales with both the number of seeds and the number of streamer heads. It is therefore preferable to keep the number of heads to a minimum. Superfluous heads are trimmed according to the criteria in (7). Then, the electrostatic shielding is set for the trimmed structure.

Update ROI. Seeds that have moved behind the ROI, collided with the streamer, or lead to a critical avalanche are removed and replaced by a new seed. When a seed is replaced, the new seed is placed a distance, equal to the height of the ROI, closer to the plane, at a random xy-position within the ROI radius. If the streamer has moved closer to the plane, the ROI moves as well, and seeds behind the new position are replaced. If a streamer head is close to the edge of the ROI, the ROI expands towards the maximum radius. New seeds are created at random position within the expanded region.

Evaluate iteration. Iteration data is extracted from the various classes to be saved for later use. The data is used to evaluate whether any stop condition is fulfilled, and stored to dedicated classes. Which data to store and how often to sample the data is controlled by the user input. The saved data is dumped to a file at regular intervals to keep memory requirements of the program low. Information to monitor the progress is printed to the screen, at regular intervals. Finally, a number of temporary variables, relevant only to the iteration is cleared, and the program is prepared for a new iteration.

Terminate? If none of the conditions for stopping a simulation are met, the next iteration is performed. These conditions include low streamer speed, streamer head close to the plane (breakdown), simulation time exceeded, computation time exceeded, and more. When a criterion is met, any unsaved data is dumped to disk, and a final logging to file and screen is performed, before the program terminates.

Appendix B. Parameters

The parameters for a simulation are supplied by the user in a JSON-formatted file (see Listing 1). A list of all important parameters is shown in Table 1. Most of the default parameter values were motivated, calculated, and/or tested for sensitivity in [22], whereas [23] and [24] introduced a some new parameters and justified their default values.

Experimental conditions. The potential, position and size of the needle are important parameters in an experiment. These parameters give the origin and the maximum potential of the streamer in the simulations. We have mostly dealt with streamers in small gaps $d_g = 3$ mm to 10 mm at potentials $V_n = 30$ kV to 150 kV [22–24], however, single-channel streamers in gaps up to 50 mm have also been simulated [42]. The potential is perhaps the most important parameter of a simulation, and as such, great care should be taken when selecting its value. Higher potentials can, for instance, require a larger ROI and smaller time steps. More details follow below.

Seeds and avalanches. The creation and movement of seeds (anions, electrons, and avalanches), as well as the growth of the avalanches, are controlled by the parameters of the liquid. We have based the simulations on cyclohexane as a model liquid since most of its properties are well-known [22]. The number of seeds in a simulation is given by the ROI volume and the seed density. The latter property is calculated from the low-field conductivity and the ion mobility, unless explicitly set by the user [22]. The remainder of the liquid parameters relates mainly to the threshold for electron detachment, electron avalanche growth, and critical avalanche size, see (2) to (6). Further information on electron avalanches and their parameters is found in e.g. [22,30,31,43,44].

Streamer structure. The parameters for the streamer can be split in two groups. The first group controls creation of the streamer heads and how they are treated in relation to each other, whereas the second group is related to the RC-model, controlling the potential of the streamer heads and the electric field within the streamer channel. There is also the option to choose whether the electric field from the streamer, acting on the seeds, is calculated using 32- or 64-bit precision. The latter requires about twice the time to compute. The tip radius of the streamer heads is chosen based on the inception of second-mode streamers [22], and this parameter is essential to calculate the electric field. In this respect, d_m and k_c, see (7), are also important. The first one sets the minimum distance between two heads, and the latter, the minimum scaling of a head. A low d_m allows for a fine-branched streamer, whereas higher values can suppress branching altogether. For higher potentials, it makes sense to reduce k_c, allowing more streamer heads to be kept during the simulation. The minimum potential difference between the needle and each streamer head is given by the field within the channel E_s [22]. The difference can be larger if the conductance is low, however, it can be limited to a maximum by the value of E_{bd} [23]. It is also possible to change the way the resistance and the capacitance is calculated, for instance as parallel plane or sphere capacitor, but this has insofar only been used to explore differences in outcomes. The time constant τ_0 (in (8)) and E_{bd} are the actual main contributors of the RC-model and are of importance to the streamer breakdown simulations [23]. Furthermore, the parameters for photoionization are included in (10) to increase the speed when a given threshold field is exceeded [24].

Simulation algorithm. The parameters in the last section are mainly related to the simulation algorithm itself. The time step and the number of steps per loop, limiting the maximum movement of seeds, are essential for a good results. The random seed, used to initialize the random number engine, controls the initial placement of seed anions. Choosing the same random number enables the study of, for instance, changing voltage with the same initial seed configuration. Conversely, not setting the random number makes the simulations uncorrelated, which is better for statistics. The size of the ROI decides how many seeds that are included in a simulation. For instance, an ROI of 10 mm^3 in combination with

Table 1
Main parameters for simulation program. The *experimental conditions* specify an overvoltage applied to medium size needle–plane gap. The values of the physical parameters in *seeds and avalanches* and *streamer structure* are justified in our previous work [22–24]. Parameter values related to the *simulation algorithm*, or the model in general, have also been discussed in previous work. See further description in Appendix B.

Property	Keyword	Symbol	Default
Experimental conditions			
Distance from needle to plane	`gap_size`	d_g	10 mm
Voltage applied to needle	`needle_voltage`	V_n	100 kV
Needle tip radius	`needle_radius`	r_n	6.0 µm
Seeds and avalanches			
Seed number density	`seeds_cn`	n_{seeds}	2×10^{12} /m^3
Anion mobility	`liquid_mu_ion`	μ_{ion}	0.30 mm^2/Vs
Electron mobility	`liquid_mu_e`	μ_e	45 mm^2/Vs
Liquid low-field conductivity	`liquid_sigma`	σ_{ion}	0.20 pS/m
Electron detachment threshold	`liquid_Ed_ion`	E_d	1.0 MV/m
Growth calculation method	`alphakind`	–	`I2009`
Critical avalanche threshold	`Q_crit`	Q_c	23
Electron multiplication threshold	`liquid_Ec_ava`	E_c	0.2 GV/m
Electron scattering constant	`liquid_Ealpha`	E_α	1.9 GV/m
Max avalanche growth	`liquid_alphamax`	α_m	130 µm^{-1}
Additive IP diff. factor	`liquid_k1`	k_α	2.8 eV^{-1}
Base liquid IP	`liquid_IP`	I_b	10.2 eV
Additive IP	`additive_IP`	I_a	7.1 eV
Additive mole fraction	`additive_cn`	x_{add}	0.00
Streamer structure			
Streamer head tip radius	`streamer_head_radius`	r_s	6.0 µm
Minimum field in streamer channel	`streamer_U_grad`	E_s	2.0 kV/mm
Streamer head merge distance	`streamer_d_merge`	d_m	25 µm
Electrostatic shielding threshold	`streamer_scale_tvl`	k_c	0.20
Photoionization threshold field	`streamer_photo_efield`	E_w	3.1 GV/m
Photoionization added speed	`streamer_photo_speed`	v_w	20 km/s
Data type for field calculation	`efield_dtype`	–	`float32`
RC-model time constant	`rc_tau0`	τ_0	1 µs
RC resistance model	`rc_resistance`	–	`linear`
RC capacitance model	`rc_capacitance`	–	`hyperbole`
RC breakdown field	`rc_breakdown`	E_{bd}	6 kV/mm
Simulation algorithm			
Time step of avalanche movement	`time_step`	Δt	1.0 ps
Max avalanche steps per iteration	`micro_step_no`	N_{MSN}	100
Seed for random number generator	`random_seed`	–	`None`
Number of similar simulations	`simulation_runs`	–	10
ROI – behind leading head	`roi_dz_above`	z^+_{ROI}	1.0 mm
ROI – in front of leading head	`roi_dz_below`	z^-_{ROI}	1.0 mm
ROI – radius from center	`roi_r_max`	r_{ROI}	3.0 mm
Stop – low streamer speed	`stop_speed_avg`	v_{min}	100 m/s
Stop – streamer close to plane	`stop_z_min`	z_{min}	50 µm
Stop – avalanche time	`stop_time_since_avalanche`	t^{ava}_{max}	100 ns
Sequential start number	`seq_start_no`	–	0
Enable profiling of code	`profiler_enabled`	–	False
Interval – dump save data to file	`file_dump_interv`	–	500
Interval – display data on screen	`display_interv`	–	500
Level of logging to file	`log_level_file`	–	20
Level of logging to console	`log_level_console`	–	20

a seed density of 2×10^{12} /m^3 results in 20 000 seeds generated, a size which is easily treated by most computers. The size of the ROI should depend on the magnitude of the electric field [22]. A streamer branch lagging behind the ROI will die, and the ROI radius gives the maximum lateral movement of the streamer. Finally, several parameters can be used to control how long a simulation will run, to prevent wasting computational time on uninteresting simulations, and to stop a simulation if the streamer stops or reaches the other electrode. Additional parameters control how often information is logged, and how detailed the logging should be.

Appendix C. Example files

This appendix contains a number of examples of possible simulations. The files in Listings 2 to 5 are all included in the folder `examples` on GitHub [26].

Listing 2: The example file `small_gap.json` specifies a small gap and a range of voltages along with many parameter values equal to the defaults (cf. Table 1). Although all values used in a simulation are stored in the log, it is nice to be explicit in the input as well. By specifying 10 `simulation_runs` and 10 voltages, a total of 100 simulations is created from this file. Each simulation initiated with the seeds at uncorrelated positions since `random_seed` is `none`.

Listing 3: The example file `small_gap_mod.json` builds on Listing 2, but a number of parameters are modified, notably the seed density and the electron avalanche parameters, as well as several parameters for the streamer. All data is stored every 5th percent of propagation by `gp5` and every 100th nanosecond by `ta07`. Storing the properties of tens of thousands of seeds enables plotting of the development of seeds, but also requires a lot of disk space. Specifying `streamer` saves all the streamer heads every 0.1 percent of propagation and is used to evaluate the development of the streamer. Since `random_seed` is 1 and `simulation_runs` is 10, a range of `random_seed` from 1

Listing 2: The example file `small_gap.json` specifies simulations similar to the baseline studies in section 3.1 in [22].

```
{   "gap_size": 0.003,
    "needle_radius": 6e-06,
    "needle_voltage": [30e3, 40e3, 50e3, 60e3, 70e3,
                       80e3, 90e3, 100e3, 110e3, 120e3],
    "liquid_name": "cyclohexane",
    "liquid_alphamax": 200e06,
    "liquid_Ealpha": 3.0e09,
    "seeds_cn": 2e12,
    "Q_crit": 23,
    "streamer_head_radius": 6e-06,
    "streamer_U_grad": 2e+06,
    "streamer_d_merge": 50e-6,
    "streamer_scale_tvl": 0.2,
    "streamer_photo_enabled": false,
    "random_seed": null,
    "simulation_runs": 10,
    "stop_sim_time": 100e-06,
    "stop_time_since_avalanche": 1e-07,
    "stop_speed_avg": 100}
```

Listing 3: The example file `small_gap_mod.json` specifies simulations similar to the attempts to facilitate branching in section 3.5 in [22].

```
{   "gap_size": 0.003,
    "needle_radius": 6e-06,
    "needle_voltage": "linspace(30e3, 120e3, 10)",
    "liquid_name": "cyclohexane",
    "liquid_alphamax": 130e06,
    "liquid_Ealpha": 1.9e09,
    "seeds_cn": 8e12,
    "streamer_head_radius": 3e-06,
    "streamer_U_grad": 8e+06,
    "streamer_d_merge": 12.5e-6,
    "streamer_scale_tvl": 0.1,
    "streamer_photo_enabled": false,
    "random_seed": 1,
    "simulation_runs": 10,
    "save_specs_enabled": {
        "stat": true,
        "streamer": true,
        "gp5": true,
        "ta07": true}}
```

Listing 4: The example file `rc.json` specifies simulations similar to those performed in section 4 in [23].

```
{   "gap_size": 0.003,
    "needle_radius": 6e-06,
    "needle_voltage": "linspace(30e3, 120e3, 10)",
    "liquid_name": "cyclohexane",
    "liquid_alphamax": 130e06,
    "liquid_Ealpha": 1.9e09,
    "streamer_head_radius": 6e-06,
    "streamer_U_grad": 0,
    "streamer_d_merge": 50e-06,
    "streamer_scale_tvl": 0.1,
    "rc_tau0": [1e-8, 1e-4],
    "rc_resistance": "linear",
    "rc_capacitance": "hyperbole",
    "rc_breakdown": [0, 4e6, 8e6, 16e6, 64e6],
    "random_seed": null,
    "save_specs_enabled": {
        "streamer": true,
        "stat": false}}
```

Listing 5: The example file `pi.json` specifies simulations similar to those performed in section 7 in [24].

```
{   "user_comment": "Photoionization and breakdown",
    "gap_size": 0.010,
    "needle_radius": 6e-06,
    "needle_voltage": "linspace(60e3, 150e3, 100)",
    "liquid_alphamax": 130e06,
    "liquid_Ealpha": 1.9e09,
    "seeds_cn": 2e+12,
    "streamer_U_grad": 0,
    "streamer_d_merge": 50e-06,
    "streamer_scale_tvl": 0.1,
    "streamer_photo_enabled": [false, true],
    "streamer_photo_efield": 3.1e9,
    "streamer_photo_speed": 20e3,
    "rc_tau0": 1e-6,
    "rc_breakdown": [0e6, 4e6, 64e6],
    "simulation_runs": 1,
    "random_seed": null,
    "file_dump_interv": 500,
    "display_interv": 500,
    "save_specs_enabled": {
        "streamer": true,
        "stat": false}}
```

through 10 is generated, giving the same 10 initial seed positions for each voltage.

Listing 4: The example file `rc.json` specifies the low and high conductivity within the streamer channel, and a range of threshold fields for electric breakdown in the streamer channel. The `linspace`-command is a convenient method for creating a list of values. By combining 2 values for the conductivity and 5 values for breakdown with 10 values for the voltage, a total of 100 simulations are created from this file.

Listing 5: The example file `pi.json` specifies electrical breakdown in the streamer channel, with and without photoionization enabled, the keyword `user_comment` does not affect the simulations, but can be convenient to set. 600 input files are created when expanding this file (100 voltages, with and without photoionization, for three different breakdown thresholds).

References

[1] P. Wedin, IEEE Electr. Insul. Mag. 30 (2014) 20–25, https://doi.org/10.1109/MEI.2014.6943430.
[2] O. Lesaint, J. Phys. D, Appl. Phys. 49 (2016) 144001, https://doi.org/10.1088/0022-3727/49/14/144001.
[3] A. Sun, C. Huo, J. Zhuang, High Volt. 1 (2016) 74–80, https://doi.org/10.1049/hve.2016.0016.
[4] B. Farazmand, Br. J. Appl. Phys. 12 (1961) 251–254, https://doi.org/10.1088/0508-3443/12/5/310.
[5] L. Niemeyer, L. Pietronero, H.J. Wiesmann, Phys. Rev. Lett. 52 (1984) 1033–1036, https://doi.org/10.1103/PhysRevLett.52.1033.
[6] A.L. Kupershtok, Sov. Tech. Phys. Lett. 18 (1992) 647–649.
[7] P. Biller, in: Proc. 1993 IEEE 11th Int. Conf. Conduct. Break. Dielectr. Liq. (ICDL'93), 1993, pp. 199–203, https://doi.org/10.1109/ICDL.1993.593938.
[8] D.I. Karpov, A.L. Kupershtokh, in: Conf. Rec. 1998 IEEE Int. Symp. Electr. Insul. (Cat No98CH36239), vol. 2, 1998, pp. 607–610, https://doi.org/10.1109/ELINSL.1998.694866.
[9] A. Ershov, A. Kupershtokh, in: Proc. 1993 IEEE 11th Int. Conf. Conduct. Break Dielectr. Liq. (ICDL'93), 1993, pp. 194–198, https://doi.org/10.1109/ICDL.1993.593937.
[10] A. Kupershtokh, in: ICDL'96 12th Int. Conf. Conduct. Break. Dielectr. Liq., IEEE, 1996, pp. 210–213, https://doi.org/10.1109/ICDL.1996.565425.
[11] A. Kupershtokh, D. Karpov, in: Proc. 2002 IEEE 14th Int. Conf. Dielectr. Liq. ICDL 2002 (Cat No02CH37319), IEEE, Graz (Austria), 2002, pp. 111–114, https://doi.org/10.1109/ICDL.2002.1022706.
[12] A.L. Kupershtokh, D.I. Karpov, Tech. Phys. Lett. 32 (2006) 406–409, https://doi.org/10.1134/S1063785006050129.
[13] V. Lopatin, M. Noskov, R. Badent, K. Kist, A. Schwab, IEEE Trans. Dielectr. Electr. Insul. 5 (1998) 250–255, https://doi.org/10.1109/94.671949.
[14] M.D. Noskov, A.S. Malinovski, C.M. Cooke, K.A. Wright, A.J. Schwab, J. Appl. Phys. 92 (2002) 4926–4934, https://doi.org/10.1063/1.1506395.
[15] I. Fofana, A. Beroual, Jpn. J. Appl. Phys. 37 (1998) 2540–2547, https://doi.org/10.1143/JJAP.37.2540.
[16] J. Qian, R.P. Joshi, E. Schamiloglu, J. Gaudet, J.R. Woodworth, J. Lehr, J. Phys. D, Appl. Phys. 39 (2006) 359–369, https://doi.org/10.1088/0022-3727/39/2/018.
[17] I. Simonović, N.A. Garland, D. Bošnjaković, Z.L. Petrović, R.D. White, S. Dujko, Plasma Sources Sci. Technol. 28 (2019) 015006, https://doi.org/10.1088/1361-6595/aaf968.
[18] J. Jadidian, M. Zahn, N. Lavesson, O. Widlund, K. Borg, IEEE Trans. Plasma Sci. 42 (2014) 1216–1223, https://doi.org/10.1109/TPS.2014.2306197.
[19] G.V. Naidis, J. Phys. D, Appl. Phys. 48 (2015) 195203, https://doi.org/10.1088/0022-3727/48/19/195203.
[20] G.V. Naidis, J. Phys. D, Appl. Phys. 49 (2016) 235208, https://doi.org/10.1088/0022-3727/49/23/235208.
[21] H. Fowler, J. Devaney, J. Hagedorn, IEEE Trans. Dielectr. Electr. Insul. 10 (2003) 73–79, https://doi.org/10.1109/TDEI.2003.1176564.
[22] I. Madshaven, P.-O. Åstrand, O.L. Hestad, S. Ingebrigtsen, M. Unge, O. Hjortstam, J. Phys. Commun. 2 (2018) 105007, https://doi.org/10.1088/2399-6528/aae0e6.
[23] I. Madshaven, O.L. Hestad, M. Unge, O. Hjortstam, P.-O. Åstrand, Plasma Res. Express 1 (2019) 035014, https://doi.org/10.1088/2516-1067/ab4072.
[24] I. Madshaven, O. Hestad, M. Unge, O. Hjortstam, P. Åstrand, Plasma Res. Express 2 (2020) 015002, https://doi.org/10.1088/2516-1067/ab6320.
[25] R. Coelho, J. Debeau, J. Phys. D, Appl. Phys. 4 (1971) 1266–1280, https://doi.org/10.1088/0022-3727/4/9/305.
[26] I. Madshaven, Cerman - a software for simulation of streamers in liquids, https://github.com/madshaven/cerman.
[27] Python Software Foundation, Python Language Reference, https://www.python.org.
[28] T.E. Oliphant, Comput. Sci. Eng. 9 (2007) 10–20, https://doi.org/10.1109/MCSE.2007.58.
[29] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, et al., Nat. Methods 17 (2020) 261–272, https://doi.org/10.1038/s41592-019-0686-2.
[30] S. Ingebrigtsen, H.S. Smalø, P.-O. Åstrand, L.E. Lundgaard, IEEE Trans. Dielectr. Electr. Insul. 16 (2009) 1524–1535, https://doi.org/10.1109/TDEI.2009.5361571.
[31] V.M. Atrazhev, E.G. Dmitriev, I.T. Iakubov, IEEE Trans. Electr. Insul. 26 (1991) 586–591, https://doi.org/10.1109/14.83675.
[32] A. Pedersen, IEEE Trans. Electr. Insul. 24 (1989) 721–739, https://doi.org/10.1109/14.42156.
[33] D. Linhjell, L. Lundgaard, G. Berg, IEEE Trans. Dielectr. Electr. Insul. 1 (1994) 447–458, https://doi.org/10.1109/94.300288.
[34] O. Lesaint, M. Jung, J. Phys. D, Appl. Phys. 33 (2000) 1360–1368, https://doi.org/10.1088/0022-3727/33/11/315.
[35] D. Linhjell, S. Ingebrigtsen, L. Lundgaard, M. Unge, in: Proc. Nord Insul. Symp., 2013, pp. 191–196, https://doi.org/10.5324/nordis.v0i23.2488.
[36] D. Linhjell, L. Lundgaard, M. Unge, O. Hjortstam, J. Phys. Commun. (2020), https://doi.org/10.1088/2399-6528/ab7b31.
[37] D. Linhjell, L.E. Lundgaard, M. Unge, in: 2019 IEEE 20th Int. Conf. Dielectr. Liq., 2019-June, 2019, pp. 1–3, https://doi.org/10.1109/ICDL.2019.8796570.

[38] H.S. Smalø, O.L. Hestad, S. Ingebrigtsen, P.-O. Åstrand, J. Appl. Phys. 109 (2011) 073306, https://doi.org/10.1063/1.3562139.
[39] G. Massala, O. Lesaint, IEEE Trans. Dielectr. Electr. Insul. 5 (1998) 371–380, https://doi.org/10.1109/94.689426.
[40] O. Lesaint, L. Costeanu, IEEE Trans. Dielectr. Electr. Insul. 25 (2018) 1949–1957, https://doi.org/10.1109/TDEI.2018.007419.
[41] S. Boggs, IEEE Trans. Dielectr. Electr. Insul. 12 (2005) 929–938, https://doi.org/10.1109/TDEI.2005.1522187.
[42] I. Madshaven, P.-O. Åstrand, O.L. Hestad, M. Unge, O. Hjortstam, in: 2017 IEEE 19th Int. Conf. Dielectr. Liq., 2017, pp. 1–4, https://doi.org/10.1109/ICDL.2017.8124641.
[43] M. Haidara, A. Denat, IEEE Trans. Electr. Insul. 26 (1991) 592–597, https://doi.org/10.1109/14.83676.
[44] G.V. Naidis, IEEE Trans. Dielectr. Electr. Insul. 22 (2015) 2428–2432, https://doi.org/10.1109/TDEI.2015.004829.