Facharbeit

im Fach

Physik (Leistungskurs)

Thema: Computersimulation zu Blitzformen

Inhalt

1 Einleitung und Fragestellung

Der Blitz ist für die Menschheit schon seit jeher eines der eindrucksvollsten Naturphänomene. Seine Ästhetik und seine symbolische Verankerung in der menschlichen Kultur (z.B. als göttliches Zeichen) haben dazu geführt, dass er auch in der bildenden und darstellenden Kunst vielfach Verwendung findet.
Da ein Blitz, wie er in der Natur auftritt, sich nicht einfach und ohne größere Hilfsmittel erzeugen lässt, ist man schon frühzeitig dazu übergegangen, ihn für visuelle Medien (Filme, Computerspiele) im Computer zu simulieren. Dies gestaltet sich allerdings nicht immer ohne Aufwände, da die Entstehung eines Blitzes ein komplexer Prozess ist und sein Aufbau und seine Gestalt von sehr vielen Faktoren abhängen, was die Simulation von realistischen Blitzen zum Teil sehr kompliziert machen kann.
Ziel dieser Arbeit ist die Untersuchung von bestehenden Blitzsimulationen und darauf basierend die Erstellung einer eigenen Simulation eines Blitzes mit Hilfe von Computeralgorithmen und unter ausschließlicher Verwendung von Schulmathematik. Der Schwerpunkt der Simulationen liegt dabei auf der geometrischen Darstellung eines naturtypischen Blitzes aus einer Gewitterwolke zum Boden, in seiner zeitlichen und räumlichen Ausbreitung.
Wie sich an den behandelten existierenden Simulationen zeigen wird, ist neben einem Verständnis für die typische Form des Blitzes es auch notwendig, die physikalischen Grundlagen zu verstehen, die zu dieser Form führen, um diese Bedingungen ggf. im Computer nachzubilden oder zumindest grafisch zu simulieren.

2 Die Form des Blitzes

Zunächst wollen wir die Form des Blitzes untersuchen und wie diese entsteht. Um die geometrische Struktur des Blitzes möglichst gut verstehen zu können, ist es hilfreich, die physikalischen Bedingungen zu untersuchen, unter denen ein Blitz entsteht.

2.1 Blitzentstehung

Betrachtet man einen Blitz, der mit einer Hochgeschwindigkeitskamera aufgenommen wurde in Zeitlupe (15), so erkennt man, dass ein Blitz nicht einfach als plötzlicher Stromstoß aus dem Nichts erscheint, sondern, dass sich aus der Wolke

kommend zunächst ein (schwach) leuchtender Kanal mit Verästelungen aufbaut. Erst wenn der unterste Ausläufer dieser Struktur den Boden erreicht hat, bildet sich daraus die (gezackte) Linie, die man aufgrund ihrer Helligkeit mit bloßem Auge sehr gut sieht.
Diese beobachtete Entstehung des Blitzes lässt sich wie folgt in mehreren Phasen einteilen und physikalisch beschreiben.

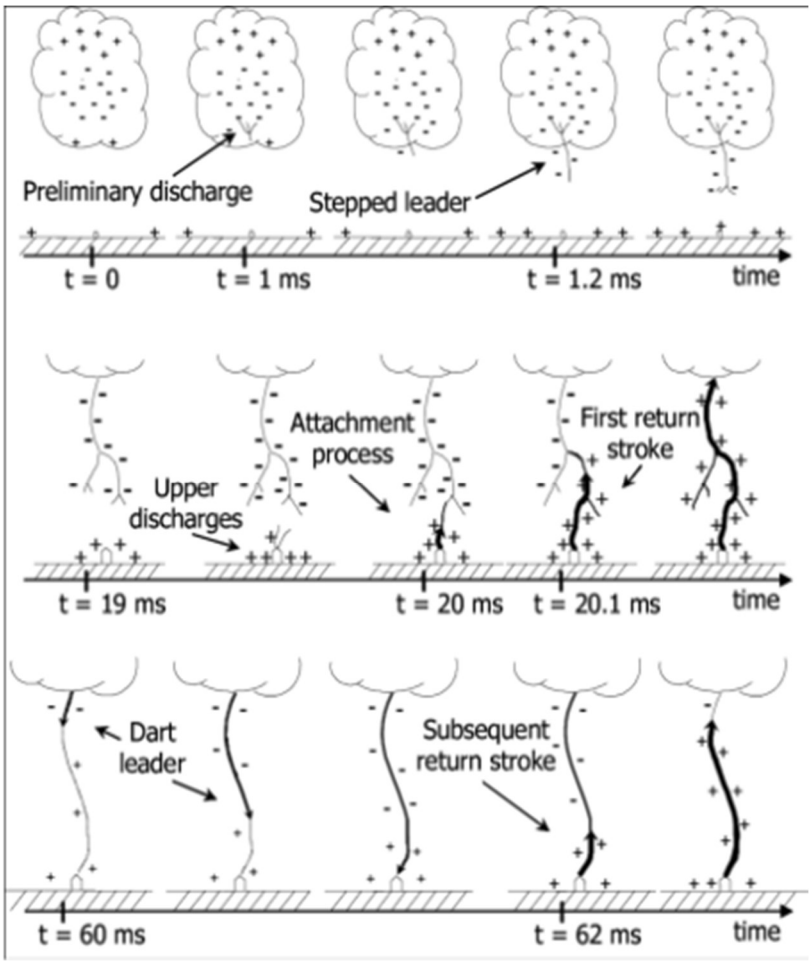


Abbildung 1: Leitblitzentstehung und Hauptentladung[1]

2.1.1 Ladungstrennung in einer Wolke

Für die Entstehung eines Blitzes muss zuerst eine Ladungstrennung innerhalb der Wolke stattfinden, bei dieser bilden sich dann zwei Ladungszentren wobei meist oben in der Wolke sich ein positives und unten in der Wolke ein negatives

[1] Abbildung aus (11)

Ladungszentrum bildet, der genaue Mechanismus für die Ladungstrennung in der Wolke ist bis heute nicht vollständig geklärt.[2]
Nach der Ladungstrennung ähnelt die Wolke annäherungsweise einem Dipol.[3] Dabei entsteht auch ein elektrisches Feld zwischen Erde und Wolke, dass dem permanenten sogenannten *Schönwetterfeld* entgegengerichtet ist.[4]
Dieses Feld ist lokal wesentlich stärker als das Schönwetterfeld und steigt so lange an, bis die Feldstärke einen kritischen Wert übersteigt. Sobald dieser Wert erreicht wurde, machen sich, ausgelöst durch eine Ionisierungslawine (Preliminary Discharge), negative Ladungsträger in der Form des *Leitblitzes* auf den Weg in Richtung Erde.[5]

2.1.2 Leitblitz und Plasmakanal

Dieser Leitblitz (Stepped Leader) bewegt sich mit einer mittleren Geschwindigkeit von 1/20 der Lichtgeschwindigkeit auf die Erdoberfläche zu, hierbei hinterlässt er einen dünnen ionisierten *Plasmakanal*, bei diesem entstehen auch die bekannten Verästelungen.[6]

Der Leitblitz folgt immer der maximalen Feldstärke, wenn dieser nah genug an der Erdoberfläche ist, kommt es zur sogenannten Fangentladung hierbei kommt dem Leitblitz eine *Fangentladung* entgegen.[7]

2.1.3 Die Fangentladung

Eine Fangentladung (Upper Discharge) besteht aus positiven Ladungsträgern, die sich wegen der lokal steigenden Feldstärke nahe der Oberfläche angesammelt haben.

Fangentladungen gehen meist von erhöhten Punkten aus, da dort die Feldstärke maximal ist, dabei breitet sich Fangentladung mit einer Geschwindigkeit von bis zu 10^6 m/s aus.[8]

---

[2] S. (2)
[3] Ebd.
[4] S. (3)
[5] Ebd.
[6] Ebd.
[7] Ebd.
[8] Vgl. (1) S. 101

2.1.4 Hauptentladung

Sobald also ein vollständig ionisierter Plasmakanal zwischen Erde und Wolke vorliegt, beginnt die *Hauptendladung* der sogenannte „Return Stroke“, dieser fließt durch den dünnen ionisierten Kanal, der dadurch noch mehr erhitzt wird und dadurch noch leitfähiger wird, und kann dabei eine Stromstärke von bis zu 100kA und eine Temperatur von bis zu 30000°C erreichen. Durch die Erhitzung des Kanals, dehnt sich das Plasma aus, wodurch ein lauter Knall (der sogenannte Donner[9]) erzeugt wird.[10]

Manchmal schlägt der Blitz in derselben Form mehrfach ein, man spricht dann von einer *Folgeentladung*.

2.1.5 Folgeentladung

Durch den Hauptblitz, wird meistens der Hauptteil der Ladung abgeführt, manchmal jedoch verbleibt genügend Ladung für weitere Entladungen in der Wolke, dann gibt es sogenannte *Folgeblitze*. Bei diesen Folgeblitzen kommt ein erneuter Leitblitz (Dart Leader) durch denselben Kanal. Dieser Leitblitz bewegt sich im Vergleich zum ersten Leitblitz mit einer konstanten Geschwindigkeit, wenn dieser den Boden erreicht, erfolgt ein sogenannter Folgeblitz. Dieses Phänomen kann sich mehrfach wiederholen, Messdaten aus Florida ergeben eine mittlere Anzahl von 4,6 Entladungen pro Blitz.[11]

2.1.6 Blitzarten

Es gibt sechs verschiedene Blitzarten, die sich bezüglich ihrer Ladungsrichtung (Übertragung von positiven oder negativen Ladungen) und ihrer Ausgangs- und Einschlagspunkte (Wolke, Boden) unterscheiden: der Wolke-Wolke Blitz, der „intra-cloud“ Blitz, der negative Wolke-Erde Blitz, der positive Wolke-Erde Blitz, der negative Erde-Wolke Blitz und der positive Erde-Wolke Blitz. Diese Arbeit behandelt nur den negativen Wolke-Erde Blitz, da diese Art 90% der Blitze zwischen Wolken und

[9] S. (1) S.122
[10] S. (3)
[11] S. (2)

der Erde ausmacht und die Blitzentstehungen bei allen Blitzarten ähnlich verläuft, weshalb ihre Form weitgehend ähnlich ist.[12]

2.2 Blitz als diskontinuierliche Entladung

Woher kommt die Konzentration der Energie beim Blitz: Die Wolke entlädt sich in Form eines Blitzes, aber warum nicht als „Ladungsregen“, d.h. als kontinuierlicher Ladungsniedergang auf breiter Fläche?
Das Hindernis für eine kontinuierliche Entladung der Wolke ist, dass die Luft zwischen Wolke und Erde ein Isolator ist.
Die isolierende Eigenschaft der Luft wird wie oben beschrieben umgangen, indem sich durch die hohe Feldstärke ein Leitblitz aufbaut, der durch seine starke Hitze den Plasmakanal erzeugt.
Plasma ist im Gegensatz zu Luft ein idealer Leiter und erlaubt damit die Hauptentladung, bei der im Vergleich zum Leitblitz der eigentliche Ladungsübertrag stattfindet.[13]
Obwohl der Plasmakanal als Leiter eigentlich eine kontinuierliche Entladung erlaubt, bleibt er nur so lange bestehen, wie er durch den Entladungsstrom stark genug erhitzt wird. Sinkt hingegen die Temperatur unter einen kritischen Wert, wird die Luft wieder zu einem Isolator und der Plasmakanal kollabiert.[14]

2.3 Gezackte Form

Die Form eines Blitzes kann variieren, da der Leitblitz immer der maximalen Feldstärke folgt, diese wiederum wird beeinflusst von mehreren Faktoren: Z.B. Der Luftfeuchtigkeit, der Luftdichte und der Lufttemperatur. Dies heißt aber nicht, dass es unmöglich ist, dass der Blitz eine gerade Linie ist, man kann dies sogar steuern. Dies schafft man, indem man nicht den Leitblitz benutzt, um den Plasmakanal zu erzeugen, sondern stattdessen einen Laser nutzt und damit in die Wolke leuchtet, dies erzeugt durch die Erhitzung der Luft einen geraden Plasmakanal, der sogar noch kontinuierlich aufrechterhalten werden kann, solange der Laser nicht ausgeschaltet wird. Dadurch kann man Blitze gezielt einschlagen lassen.[15]

---

[12] S. (3)
[13] S. (7)
[14] S. (1) S. 122
[15] S. (8), (9)

2.4 Verästelung und Selbstähnlichkeit

Die Verästelungen des Blitzes entstehen nicht einfach rein zufällig, sondern folgen dem Prinzip der *Selbstähnlichkeit*[16]: Die Verästelungen sehen selbst wieder, wie kleine Blitze aus, und wenn man in den Blitz mit einer Lupe immer weiter „hineinzoomen“ könnte, würde man immer wieder auf eine ähnliche Form der Verästelungen treffen. Dieses Verhalten, das man auch *fraktal* nennt, tritt bei den bei dem unterschiedlichsten physikalischen Phänomen auf.[17] So wird z.B. in (7) berichtet, dass die Verästelung des Blitzes, der gleicht, die beim Vermischen zweier Flüssigkeiten unterschiedlicher Viskosität entsteht. Beim Blitz entstehen diese immer, wenn der Plasmakanal zwischenzeitlich zum idealen Leiter wird, da dabei die Grenze zwischen dem Plasma und der Luft instabil wird und dadurch sich der Blitz verzweigen kann. Hierbei sind die beiden „Medien“ mit verschiedener Viskosität zum einen die ionisierte Luft (Plasma) und die neutrale Luft.[18]
Das Prinzip der Selbstähnlichkeit lässt sich in der Computersimulation sehr gut mit Hilfe des Konzeptes der *Rekursion* umsetzen. Wie wir in den Beispielsimulationen weiter unten sehen werden, werden dort rekursive Verfahren auch vielfach eingesetzt. Eine Simulation, (11), verwendet sogar einen rekursiven Algorithmus, der aus der Botanik stammt, um Blitze darzustellen.

3 Simulation von Blitzen

3.1 Kategorisierung bestehender Blitzsimulationen

Betrachtet man die vorliegenden Blitzsimulationen (einen guten Überblick liefert z.B. (11)), so scheint eine Unterscheidung der Blitzsimulationen in zwei Kategorien sinnvoll:

- Physikalisch fundierte Ansätze versuchen die physikalischen Zusammenhänge und die dazugehörigen mathematischen Gleichungen möglichst genau umzusetzen und vereinfachen nur die aufwändigsten Schritte der Berechnung

[16] S. (16)
[17] S. (17)
[18] S. (7)

- Phänomenologische Ansätze hingegen gehen vor allem von der Geometrie des Blitzes aus und versuchen, diese weitgehend unabhängig von der zugrundeliegenden Physik nachzubilden

Um meinen eigenen Algorithmus zu schreiben, habe ich mir zuerst 4 verschiedene Blitzsimulationen und deren Algorithmen angeschaut. Zwei physikalisch fundierte: (6) und (10), und zwei phänomenologische (5) und (11).

3.1.1 Physikalisch fundierte Blitzsimulation

Die physikalisch fundierten Blitzsimulationen teilen den Raum in kleine gleichgroße Zellen ein und berechnen für diese Zellen jeweils mit Hilfe der physikalischen Formeln die zugehörigen physikalischen Größen (Spannung bzw. Potential und/oder Strom).

3.1.1.1 Dielectric Breakdown Model

Bei diesem Ansatz, der in (6), (10), (12) und (14) erwähnt wird, wird der Raum in Gitterzellen aufgeteilt, in denen sich der Leitblitz ausbreitet. Es ergibt sich der jeweils nächste Schritt des Leitblitzes durch:

- die Ermittlung des Potentials mit Hilfe der Lösung der Laplace Gleichung
- die zufällige Auswahl einer Nachbarzelle als neue Leitblitzzelle (d.h. Setzen des Potentials dieser Zelle auf null gegenüber der Wolke), des bereits vorhanden Blitzes gewichtet mit dem errechneten Potenzial
- Nachbarzellen sind hierbei alle Zellen, die an die bereits ermittelten Leitblitzzellen angrenzen, dadurch können Verästelungen entstehen.
- Wenn der Leitblitz die Erde erreicht, findet die Entladung entlang des kürzesten Weges innerhalb der ermittelten Leitblitzzellen statt.

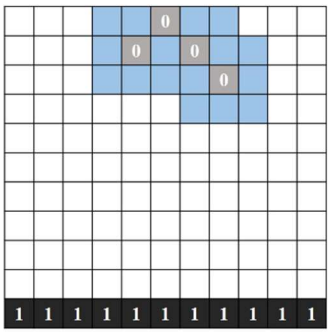


Abbildung 2: Ein Iterationsschritt aus dem in (14) beschriebenen Algorithmus: Leitblitzzellen (grau, mit Potential 0), Kandidaten für neue Leitblitzzellen (hellblau) und Erdungspotential (Wert 1)

3.1.1.2 Gitter aus Widerständen

Der Ansatz, der in (10) als „A threshold mechanism ensures minimum-path flow in lightning discharge“ beschrieben wird, berechnet die durch jede Zelle fließenden Strom und geht daher von einem mikroskopisch kleinen Gitter aus „Elementar-widerständen“ aus, die jeweils einen kapazitiven Widerstand (der Widerstand hängt von der zeitlichen Ableitung der Potenzialdifferenz ab) und einem nicht-linearen Widerstand (der Widerstand hängt von der Potenzialdifferenz selbst ab) besteht. Die Widerstandsfunktion (Φ[19] genannt) bildet den Widerstand der Luft ab; so gut wie keine Leitung bei „niedrigen“ Spannungen (d.h. weit unterhalb der Durchschlagsspannung) und gute Leitfähigkeit oberhalb der Durchschlagsspannung.

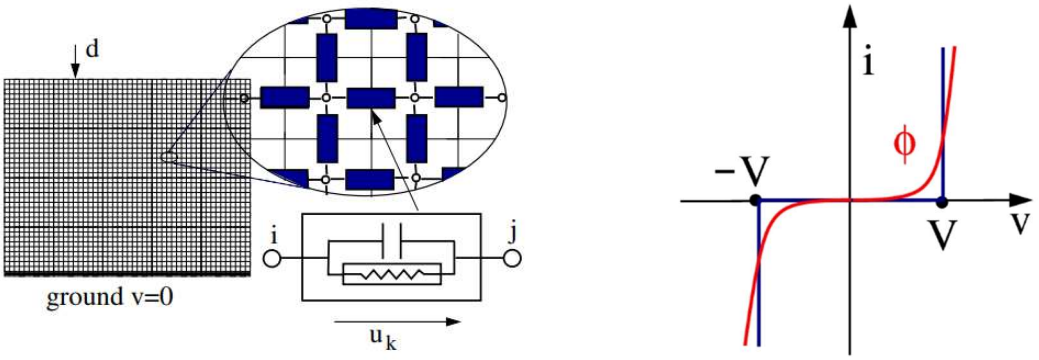


Abbildung 3: Das in (10) beschriebene Widerstandsgitter und die verwendete Widerstandsfunktion Φ

Damit kann das folgende physikalische Verhalten des Blitzes beschrieben werden:

- In der Aufbauphase spielen vor allem die kapazitiven Widerstände der einzelnen Elementarwiderstände eine Rolle, da die Potenzialänderungen durch den Aufbau des Leitblitzes gegenüber der Widerstandsfunktion dominieren (es fließt vergleichsweise wenig Strom).

[19] Nicht zu verwechseln mit dem in der Physik sonst üblicherweise mit Φ bezeichnetem Potenzial!

- In der Haupt- und Folgeentladungsphase wird der kapazitive Widerstand im Vergleich zum nicht-linearen Widerstand vernachlässigt werden, da im Rahmen der kontinuierlichen Entladung die Potenzialänderung über den Plasmakanal gleichverteilt ist (es fließt ein großer und konstanter Strom).

$$i_k = \phi_k \left(v_h - v_j\right) + \frac{d}{dt}\left[C_k(v_h - v_j)\right]$$

Abbildung 4: Verwendete Formel zur Berechnung der Stromstärke in einen einzelnen Gitterpunkt

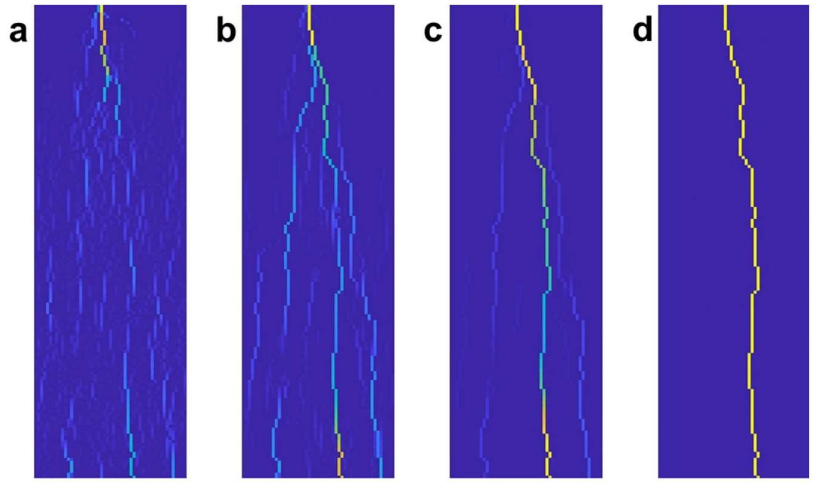


Abbildung 5: Leitblitzentstehung und Hauptblitz, erzeugt mit der Simulation aus (10)

Der in (10) formulierte Ansatz erscheint physikalisch sehr interessant, da nur ein einziges Gleichungssystem notwendig ist, um sowohl die Leitblitzentstehung als auch die Hauptentladung zu beschreiben. Allerdings scheint die grafische Darstellung (s. Abbildung 5) nicht das Hauptziel des Ansatzes darzustellen.

3.1.2 Vor- und Nachteile der physikalisch fundierten Simulation

Der große Vorteil der physikalisch fundierten Simulationen ist, dass sie universell einsetzbar sind, d.h. sie sind nicht nur für Wolke-Erde Blitze einsetzbar, sondern für alle Blitzszenarien in der Natur oder im Labor. Der große Nachteil wiederum besteht darin, dass diese Algorithmen mit Schulmathematikkenntnissen nicht verstanden werden können und auch sehr berechnungsintensiv sind (da man für das ganze Gitter die Werte berechnen muss und nicht nur für den Weg des Blitzes).

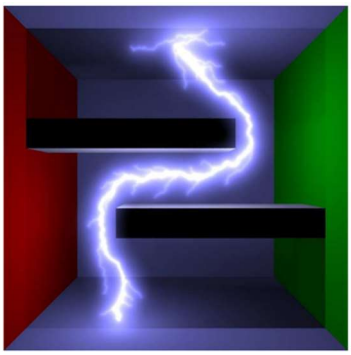


Abbildung 6: Die in (14) beschriebene physikalisch fundierte Simulation erlaubt es, zu beliebigen Randbedingungen im Raum die passende Blitzstruktur zu berechnen und darzustellen

Das Problem der hohen Komplexität tritt bei der anderen Kategorie der Blitzsimulation nicht auf, da diese Simulationen auf den physikalischen Unterbau weitgehend verzichten und sich nur auf die Darstellung des Blitzes konzentrieren.

3.1.3 Phänomenologische Blitzsimulation

3.1.3.1 Einfacher rekursiv geometrischer Algorithmus

Der Ansatz, der in (5) beschrieben wird, verwendet sogar keinerlei physikalisch begründete Formeln und nutzt einen rekursiv geometrischen Algorithmus, um die Blitze zu erzeugen:

- Initial wird ein Liniensegment zwischen dem Start- und Endpunkt des Blitzes (oberes Bildschirmende und Mitte des Bildschirms) erzeugt
- Aus diesem Liniensegment wird rekursiv durch Einfügen jeweils eines Zwischenpunktes auf halber Höhe zwischen dem Start- und Endpunkt eine weitere „Zacke“ in den Blitz eingefügt
- Dieses rekursive Einfügen erfolgt bis zur einer Rekursionstiefe von sieben, d.h. am Ende hat der Blitz $2^7 = 128$ Segmente die eine entsprechend gezackte Linie bilden
- Mit einer gewissen Chance kann sich am Ende eines Segments eine Verzweigung bilden, deren Richtung mithilfe einer Rotationsabbildung von der ursprünglichen Richtung des Segments abweicht

Dieser Ansatz ist sehr viel einfacher als die physikalischen Ansätze oben. Dadurch, dass als Datenstruktur nicht der Bildschirm als Gitter verwendet, sondern die

Liniensegmente, die zusammen den Blitz bilden, wird die Berechnung sehr viel schneller.
Das in (5) dargestellte Beispielergebnis vermag aber optisch nicht zu überzeugen:

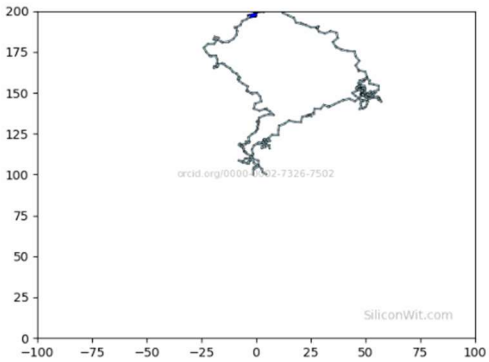


Abbildung 7: Mit (5) simulierte Blitze

- Durch den vorgegebenen Endpunkt gibt es wenig Variabilität in x-Richtung. Vielmehr muss auf eine zufällige maximale Auslenkung Δx zwangsläufig am Ende eine gegenläufige Auslenkung um -Δx erfolgen, was nicht sehr natürlich aussieht
- Die Umsetzung der Verzweigungen erscheint fehlerhaft, da keine echte Verästelungsstruktur zu erkennen ist

Insgesamt kann man sagen, dass die Umsetzung in (5) auf physikalische Gesetzmäßigkeiten zur Herleitung weitgehend verzichtet und die optischen Ergebnisse unbefriedigend bleiben. In der Hoffnung auf bessere Resultate wollen wir nun einen stärker physikalisch inspirierten Algorithmus betrachten.

3.1.3.2 Space Colonization-Algorithmus

Der Ansatz, der in (11) beschreiben wird, nutzt einen sogenannten „Space Colonization“ Algorithmus der eigentlich entwickelt wurde, um die Adern eines Blattes oder die Äste eines Baumes zu simulieren.
Die Umsetzung erfolgt über die Verwendung einer aus Einzelsegmenten zusammengesetzten Verästelungsstruktur und unter Zuhilfenahme von zufällig gestreuten Punkten als Attraktoren, die das Wachstum der natürlichen Struktur – hier der Blitz – lenken:

- In jedem Wachstumsschritt wird über alle Attraktoren iteriert und überprüft, ob es ein Segment in einer definierten Mindestnähe gibt

- Wenn dies der Fall ist, wird ein Richtungsvektor von dem Segmentende zu dem Attraktor-Punkt gebildet
- Dies erfolgt für alle hinreichend nahegelegenen Attraktoren, sodass sich durch Vektoraddition eine Gesamtrichtungsvektor ergibt, der die Richtung für das neue Segment bestimmt
- Nach hinzufügen des neuen Segments, werden alle Attraktor-Punkte entfernt, die sich in einer definierten „Berührungsnähe“ des Segmentendes befinden
- Der Algorithmus ist beendet, wenn alle Attraktor-Punkte verschwunden sind. Übrig bleibt eine verästelte Struktur
- Um diese Struktur für eine Blitzsimulation zu optimieren, werden in (11) weitere Hilfspunkte (Anfangs-, End- und Stützpunkte) gesetzt, durch die diese Struktur laufen muss. Dadurch wird eine grobe Richtungslinie für den Blitz definiert. Außerdem werden die Attraktor-Punkte nur in einem vorgegebenen Kanal rund um diese Richtungslinie gesetzt
- Da bei diesem Vorgehen die Verästelungen nur noch sehr kurz werden, werden basierend auf Zufallsentscheidungen weitere Verästelungen als “Blitze zweiter/dritter/vierter Ordnung“ an den Hauptblitz angehängt

Wie man erkennen kann, ist der grundsätzliche Space Colonization-Algorithmus nicht sehr komplex und mit Schulmathematikkenntnissen umsetzbar. Allerdings machen die in (11) genannten Zusatzschritte das Vorgehen um einiges komplizierter. Insbesondere die Tatsache, dass die Verästelungen des Blitzes nicht vom Space Colonization Algorithmus selbst herrühren (was sie könnten), sondern noch künstlich hinzugefügt werden müssen, erhöht aus meiner Sicht die Komplexität des Algorithmus‘ unnötig.
Dass die mit Hilfe dieses Algorithmus‘ konstruierten Blitze sehr realistisch aussehen, belegt die weiter oben formulierte These, dass Verästelungen in verschiedenen physikalischen Kontexten eine ähnliche Struktur aufweisen können.
Trotzdem finden sich in (11) nur in geringem Maße physikalische Begründungen für die Vorgehensweise, auch wenn die Physik am Anfang ausführlich beschrieben wird.

3.2 Beschreibung meiner eigenen Blitzsimulation

Nach der Analyse der verschiedenen Methoden der Blitzsimulation, habe ich mich bei meiner eigenen Blitzsimulation für eine phänomenologische Variante auf Basis physikalischer Überlegungen entschieden. Grund dafür ist, dass die Komplexität

einer rein physikalischen Simulation sehr hoch ist und der umsetzungstechnische Aufwand die optische Qualität nur begrenzt rechtfertigt.
Phänomenologische Ansätze hingegen sind mathematisch sehr einfach gehalten und lassen sich daher leicht programmiertechnisch umsetzen, führen aber aufgrund fehlender physikalischer Grundlagen häufig zu optisch unbefriedigenden Ergebnissen oder benötigen wie in (11) aufwändigere und wenig physikalisch begründete Erweiterungen. Deshalb habe ich versucht, einen Blitz ähnlich wie in (5) auf phänomenologischer Basis zu erzeugen, der durch Erweiterungen – ähnlich wie in (11) – aber um einiges physikalisch akkurater ist.

3.2.1 Verwendete Datenstrukturen

3.2.1.1 Polylinien als Blitzsegmente

In meinem eigenen Algorithmus nutze ich – anders als die anderen phänomenologischen Simulationen – Polylinien (d.h. eine Verbindung von mehreren geraden Linien statt einer einzelnen geraden Linie) als Segmente um den Blitz darzustellen, Polylinien sind vorteilhaft, da man diesen eine gezackte und damit blitzartige Struktur geben kann.

3.2.1.2 Rekursive Definition der Ordnung eines Segments

Ähnlich wie die anderen phänomenologischen Simulationen habe ich die Blitzsegmente in verschiedene Ordnungen eingeteilt, wobei das Segment mit dem Endpunkt, der am nächsten zum Boden liegt, immer die Ordnung null erhält. Die Segmente mit der Ordnung null bilden den sogenannten Hauptblitz, über den letztlich die Hauptentladung stattfindet. Die Abzweigung von einem Blitzsegment der Ordnung n hat die Ordnung n+1. Die Ordnung wird verwendet, um die Länge der Nebenblitze und die Abzweigungstiefe zu begrenzen.

3.2.2 Physikalische Annahmen

Die Simulation soll sowohl die Entstehung des Leitblitzes als auch die Hauptentladung möglich naturgetreu darstellen. Dazu werden folgende physikalische Annahmen getroffen:

- Es wird von einem weitgehend homogenem E-Feld (d.h. äquidistante Potenziallinien) ausgegangen. Daraus folgt, dass Δy immer ungefähr gleich

groß ist. Die Inhomogenitäten werden in diesem Fall durch Zufallsfunktionen simuliert.

- Die Abzweigungstiefe wird begrenzt, da in der Natur sehr stark verzweigte Blitze nur selten auftreten.
- Wenn ein Ende des Blitzkanals bereits um einiges näher am Boden ist, dann kommen weiter oben keine neuen Verästelungen mehr hinzu, da der Blitzkanal in Bodennähe die größte Potenzialdifferenz erzeugt und deshalb dort der Zuwachs wahrscheinlicher ist.

3.2.3 Algorithmus

Auf Basis der obigen Annahmen ist mein Algorithmus folgendermaßen aufgebaut:

1. Der Algorithmus überprüft, ob eines der Segmente den Boden erreicht hat.
2. Wenn dies der Fall ist, wird der Hauptblitz erzeugt und der Algorithmus ist zu Ende.
3. Wenn dies nicht der Fall ist, werden alle Enden der Segmente gesucht, deren Entfernung vom Blitzstartpunkt größer ist als die Hälfte der Entfernung zwischen dem Blitzstartpunkt und dem untersten Ende des Blitzes, und deren Ordnung kleiner als vier ist.
4. Danach werden bei jedem dieser Enden zufällig null bis zwei neue Segmente angehängt. Die Zufallsfunktion ist hierbei nicht linear, sondern quadratisch, d.h. niedrigere Werte (0 oder 1) werden bevorzugt.
5. Die Ausdehnung der neu angehängten Segmente in y-Richtung ist $\Delta y = 0{,}9^{n+1}$ mit der Ordnung n.
   Die Ausdehnung in x-Richtung Δx entspricht immer einer zufälligen Zahl zwischen -1 und 1, die mit Δy multipliziert wird.

3.3 Ergebnisse

Die mit Hilfe der Simulation erzeugten Blitze können bzgl. der geometrischen Gestalt mit echten Blitzen verglichen werden. Dies erfolgt mit Hilfe eines Screenshots aus der Hochgeschwindigkeitsfilmaufnahme (15):

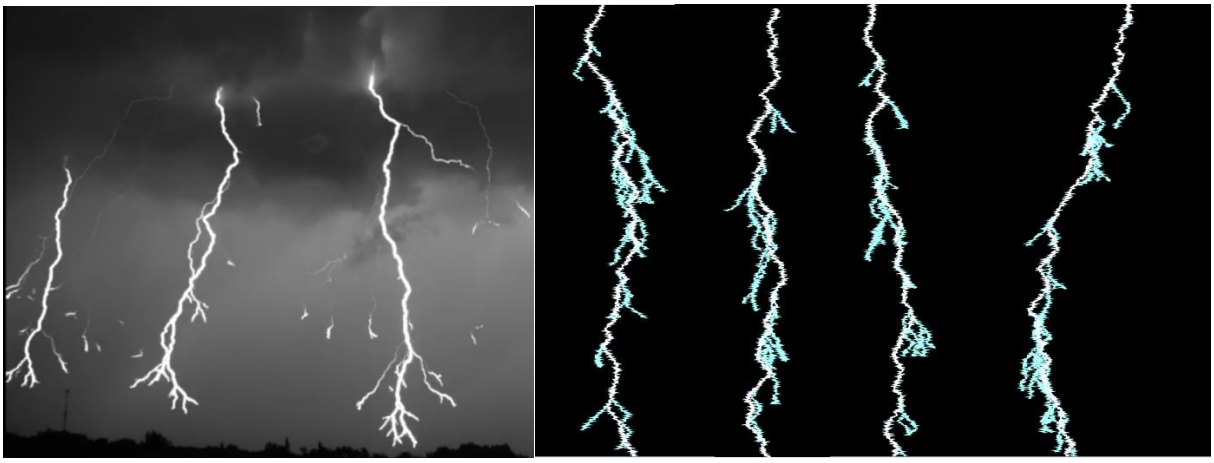


Abbildung 8: Echte Blitze (aus (15), nur Leitblitze) aus und simulierte Blitze (hellblau: Leitblitz, weiß: Hauptblitz)

3.4 Fazit und Ausblick:

Wenn man den oben genannten selbstentwickelten Algorithmus mit den weiter oben untersuchten vergleicht, kann man sagen, dass die Eigenentwicklung die vorhandenen phänomenologischen Ansätze sowohl in einzelnen Aspekten kopiert (Verwendung von Liniensegmenten, rekursives Zeichnen mit Begrenzung der Rekursionstiefe der Verästelungen) aber auch erweitert (Wachstum und Verzweigung des Leitblitzes vor allem in den Segmenten der Ordnung null).
Gegenüber allen anderen Ansätzen hat die Eigenentwicklung als Alleinstellungsmerkmal, dass für die Berechnungen ausschließlich Schulmathematik benötigt wird (und z.B. keine Rotationsmatrizen) und dadurch – zusammen mit der Einfachheit des Algorithmus – der Code sehr kompakt gehalten wird.[20] Dieser Code ist auch anders als bei den meisten der anderen Papers frei zugänglich auf Github.[21]
Die geometrische Form des erzeugten Blitzes kann als einigermaßen realistisch betrachtet werden und wäre für Einsatzszenarien wie z.B. in einem Computerspiel sicherlich ausreichend.
Das bislang noch wenig überzeugende grafische *Rendering*[22] kann unabhängig vom Algorithmus durch Einbindung einer entsprechenden Bibliothek verbessert werden.

---

[20] Die Länge des Codes beträgt lediglich 210 Zeilen. Zum Vergleich: der Code aus (6) besteht aus fast 5000 Zeilen (s. Anhang).
[21] S. (19)
[22] S. (20)

Allgemein ist eine Weiterentwicklung der Simulation vergleichsweise einfach, da durch seine Objektorientierung der Programmcode gut strukturiert und damit leicht verständlich ist.
Eventuelle Weiterentwicklungen könnten (über die oben bereits genannte Verbesserung des Renderings) die folgenden Punkte umfassen:

- Weitere Blitzszenarien
- Erweiterung ins Dreidimensionale

Beides würde natürliche die Verwendung von fortgeschritteneren mathematischen Konstrukten erfordern.

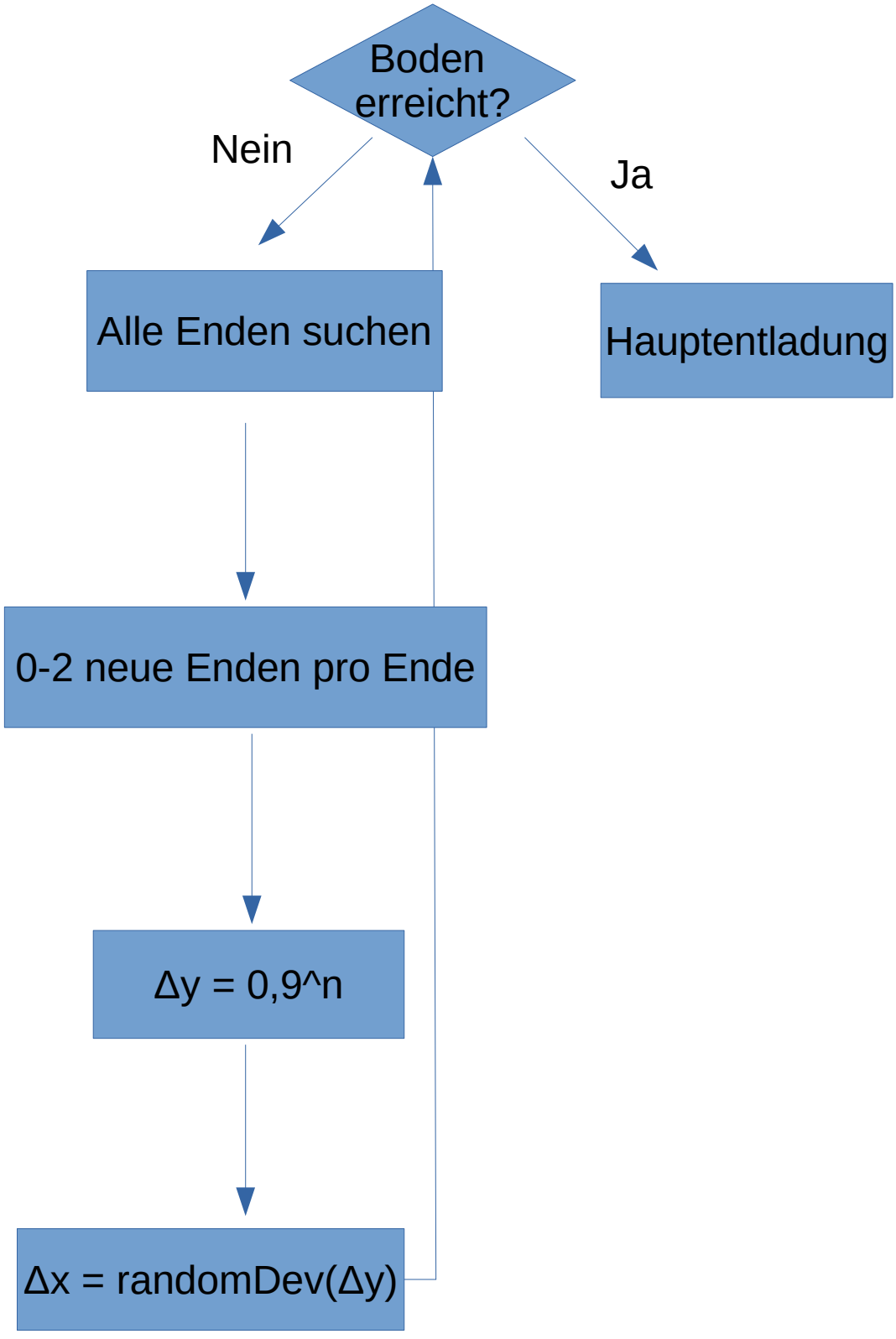
4 Quellen

1. Josef Wittmann: Physik in Wald und Flur
2. https://www.aldis.at/forschung/blitzphysik/ Zuletzt aufgerufen: 25.02.2024
3. https://www.leifiphysik.de/elektrizitaetslehre/ladungen-felder-mittelstufe/ausblick/blitze-gewittern Zuletzt aufgerufen: 25.02.2024
4. Vernon Cooray: An Introduction to Lightning
5. https://www.siliconwit.com/modelling-and-simulation-in-python/lightning-strike Zuletzt aufgerufen: 24.02.2024
6. Jeongsu Yun et al.: Physically-Inspired, Interactive Lightning Generation https://sglab.kaist.ac.kr/paper/physically_inspired_interactive_lightning_generation.pdf
7. https://www.wissenschaft.de/technik-digitales/warum-blitze-sich-verzweigen/ Zuletzt aufgerufen: 24.02.2024
8. https://www.youtube.com/watch?v=r3craaF-aFI Blitze mit dem Laser lenken Elektrizität alpha Lernen erklärt Physik (Philip probiert's) Zuletzt aufgerufen: 20.01.2024
9. https://www.zdf.de/nachrichten/panorama/blitze-laserstrahl-wissenschaft-gewitter-100.html Zuletzt aufgerufen: 20.01.2024
10. https://www.nature.com/articles/s41598-020-79463-z Zuletzt aufgerufen: 25.02.2024
11. Using a Space Colonization Algorithm for Lightning Simulation
12. Theodore Kim et al.: Fast Simulation of Laplacian Growth
13. https://www.youtube.com/watch?v=kKT0v3qhIQY Coding Challenge #17: Fractal Trees – Space Colonization Zuletzt aufgerufen: 24.02.2024
14. Theodore Kim and Ming C. Lin: Fast Animation of Lightning Using An Adaptive Mesh
15. https://www.youtube.com/watch?v=XWuZqw3LopE Lightning recorded at 7000 FPS Zuletzt aufgerufen: 25.02.2024
16. https://de.wikipedia.org/wiki/Selbst%C3%A4hnlichkeit Zuletzt aufgerufen: 25.02.2024
17. https://en.wikipedia.org/wiki/Fractal#Natural_phenomena_with_fractal_features Zuletzt aufgerufen: 25.02.2024
18. https://github.com/NSKBNick/blitz Zuletzt aufgerufen: 25.02.2024

19. https://en.wikipedia.org/wiki/Rendering_(computer_graphics) Zuletzt aufgerufen: 25.02.2024

5 Anhang

- Abbildung Algorithmus der eigenen Entwicklung
- Code Eigenentwicklung
- Tabelle: Vergleich Aller beschriebenen Blitzsimulationen


Boden erreicht?
Nein
Ja
Alle Enden suchen
Hauptentladung
0-2 neue Enden pro Ende
Δy = 0,9^n
Δx = randomDev(Δy)

```
package ui;

import geom.LightningBolt;

import javax.swing.*;
import java.awt.*;

public class DrawingPane extends JPanel {

    LightningBolt bolt = null;
    JButton button = new JButton("Blitz");

    public DrawingPane(){
        super();
        setLayout(new BorderLayout());
        button.addActionListener(e -> drawLightning());
        add(button, BorderLayout.SOUTH);
        revalidate();
        repaint();
    }

    private void drawLightning(){
        if(bolt == null || bolt.isGroundReached()){
            bolt = new LightningBolt(getRootPane().getWidth() / 2.0,0.0,
getRootPane().getHeight());
        } else {
            bolt.addRandomSegment();
            System.out.println(bolt + ": " + bolt.getYMax());
        }
        repaint();
    }

    @Override
    public void paint(Graphics g) {
        System.out.println("paint...");
        g.setColor(Color.BLACK);
        g.fillRect(0,0, this.getWidth(), this.getHeight() -
button.getHeight());

        g.setColor(new Color(179, 255, 255));
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
        Stroke stroke = new BasicStroke(1,
BasicStroke.CAP_BUTT,BasicStroke.JOIN_MITER);
        g2d.setStroke(stroke);

        if(bolt != null) {
            System.out.println("Drawing " + bolt);
            bolt.drawSteppedLeader(g2d);
            if (bolt.isGroundReached()){
                g.setColor(Color.WHITE);
                g2d.setStroke(stroke);
                bolt.drawMainBolt(g2d);
            }
        }

    }
}
```

```java
import ui.DrawingPane;

import javax.swing.*;
import java.awt.*;

public class Main {
    public static void main(String[] args) {
        //TIP Press <shortcut actionId="ShowIntentionActions"/> with your
caret at the highlighted text
        // to see how IntelliJ IDEA suggests fixing it.
        JFrame frame = new JFrame();
        frame.getContentPane().add(new DrawingPane());
        frame.setSize(640,480);
        Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
        frame.setLocation(
                dim.width / 2 - frame.getSize().width / 2,
                dim.height / 2 - frame.getSize().height / 2
        );
        frame.setVisible(true);
        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }
}
```

```
package geom;

import java.awt.*;
import java.awt.geom.Point2D;
import java.util.concurrent.ConcurrentLinkedQueue;

import static geom.BoltSegment.INTERMEDIATE_POINTS;
import static geom.BoltSegment.randomDeviate;

public class LightningBolt {

    final static public int NUM_SEGMENTS = 50;
    private final double yGround;
    private final double deltaY;
    private final ConcurrentLinkedQueue<BoltSegment> segments = new
ConcurrentLinkedQueue <>();

    public LightningBolt(double startX, double startY, double yGround){
        BoltSegment initialSegment = new BoltSegment(startX, startY,
startX, yGround / NUM_SEGMENTS, INTERMEDIATE_POINTS, 0);
        this.yGround = yGround;
        this.deltaY = yGround / NUM_SEGMENTS;
        segments.add(initialSegment);
    }

    public void drawSteppedLeader(Graphics2D g2d){
        for(BoltSegment segment: segments) {
            g2d.draw(segment.getPath());
        }
    }
    public void drawMainBolt(Graphics2D g2d){
        BoltSegment segment = getLastSegment();
        g2d.draw(segment.getPath());
        while (segment.predecessor != null) {
            segment = segment.predecessor;
            g2d.draw(segment.getPath());
        }
    }
    public void addRandomSegment(){
        for(BoltSegment segment: segments){
            int numberOfNewSegments = (int) (Math.random() * Math.random()
*2.999);
            if(segment.successor == null && segment.order < 4 &&
notToFarFromMainEnd(segment) ){
             for(int i=0; i < numberOfNewSegments; i++) {
                addNewSegment(segment);
             }
            }
        }
    }

    public boolean isGroundReached(){
        return getYMax() >= yGround;
    }

    public double getYMax(){
        double yMax = 0;
        for(BoltSegment segment: segments){
            if(segment.endPoint.y > yMax)
                yMax = segment.endPoint.y;
        }
        return yMax;
    }
```

```
    public BoltSegment getLastSegment(){
        BoltSegment lastSegment = segments.peek();
        for(BoltSegment segment: segments){
            if(segment.endPoint.y > lastSegment.endPoint.y) {
                lastSegment = segment;
            }
        }
        return lastSegment;
    }
    public String toString(){
        return "Bolt(" + segments.size() + "), maxY = " + getYMax();
    }

    private void addNewSegment(BoltSegment segment){
        int newOrder = segment.order + 1;
        Point2D.Double newStart = segment.endPoint;
        Point2D.Double newEnd = new Point2D.Double();
        newEnd.y = newStart.y + Math.pow(0.9, newOrder) * deltaY;
        newEnd.x = newStart.x + randomDeviate(deltaY);
        BoltSegment newSegment = new BoltSegment(newStart.x, newStart.y,
newEnd.x, newEnd.y, INTERMEDIATE_POINTS, newOrder);
        segment.successor = newSegment;
        newSegment.predecessor = segment;
        segments.add(newSegment);
        normalizeOrder();
    }

    private void normalizeOrder(){
        getSegmentWithYMax().order = 0;
    }

    private BoltSegment getSegmentWithYMax(){
        double yMax = getYMax();
        for(BoltSegment segment: segments){
            if(segment.endPoint.y == yMax){
                return segment;
            }
        }
        return null;  // should never be reached
    }
    private boolean notToFarFromMainEnd (BoltSegment segment){

        return   segment.endPoint.y > getYMax()/2;
    }
}
```

```
package geom;

import java.awt.geom.GeneralPath;
import java.awt.geom.Point2D;
import java.util.LinkedList;
import java.util.List;

public class BoltSegment {

    public static final int INTERMEDIATE_POINTS = 20;

    public BoltSegment successor = null;
    public BoltSegment predecessor = null;
    final private GeneralPath path;
    public int order;

    public Point2D.Double endPoint = new Point2D.Double(0.0 ,0.0);
    public BoltSegment(double startX, double startY,
                       double endX, double endY,
                       int intermediatePoints, int order){
        path = createBoltSegmentPath(startX, startY, endX, endY,
intermediatePoints);
        endPoint.x = endX;
        endPoint.y  = endY;
        this.order = order;
    }

    public GeneralPath getPath(){
        return path;
    }
    static public GeneralPath createBoltSegmentPath(double startX, double
startY,
                                                    double endX, double
endY,
                                                    int
intermediatePoints){
        List<Point2D.Double> path = new LinkedList<>();
        path.add(new Point2D.Double(startX, startY));
        path.add(new Point2D.Double(endX, endY));
        for(int i=0;i<intermediatePoints;i++) {
            path = addRandomPoint(path, i,
                    1.0 / (intermediatePoints + 1));
        }
        // remove point before last point as it has the same y coordinate
as the last point
        //path.remove(path.size() - 2);
        return pathToGeneralPath(path);
    }

    /**
     * Inserts a randomly displaced point into a given path at a given
position
     * @param path the path consisting of all points
     * @param index the position of the path, where to add the new point
     * @return the updated path (containing the newly added point at the
specified position)
     */
    static private List<Point2D.Double> addRandomPoint(List<Point2D.Double>
path,
                                                       int index, double
yDelta){
        Point2D.Double startPoint = path.getFirst();
        Point2D.Double endPoint = path.getLast();
```

```
        double newX = startPoint.x + (endPoint.x - startPoint.x) * yDelta *
(index + 1) + LightningBolt.NUM_SEGMENTS * randomDeviate(yDelta);
        double newY = startPoint.y + (endPoint.y - startPoint.y) * yDelta *
(index + 1);
        Point2D.Double newPoint = new Point2D.Double(newX, newY);
        path.add(index + 1, newPoint);
        return path;
    }

    public static double randomDeviate(double delta) {
        return  randomDev() * delta;
    }

    public static double randomDev(){
        return 1 - 2 * Math.random();
    }
    static private GeneralPath pathToGeneralPath(List<Point2D.Double>
path){
        GeneralPath generalPath = new GeneralPath();
        boolean first = true;
        for(Point2D.Double point: path){
            if(first){
                generalPath.moveTo(point.x, point.y);
                first = false;
            } else {
                generalPath.lineTo(point.x, point.y);
            }
        }
        return generalPath;
    }

    public String toString(){
        return "SEG(" + order + "): " + "( " + endPoint.x + " | " +
endPoint.y + " )" ;
    }
}
```

Algorithmus	**Verwendete Datenstruktur**	**Code**	**Code-Zeilen**	**Klassen/ Prozeduren**	**Notwendige Mathematik-Kenntnisse**	**Berechnung und Darstellung der Leitblitzentstehung**	**Realismus der erzeugten Blitzgeometrie**	**Weitere Vorteile (+) und Nachteile (-)**
Physikalisch fundiert: DBM (6), (14)	Gitter aus Potentialzellen	Objektorientiert, C++	4957	11	hoch	ja	++	+ beliebige Blitzszenarien berechenbar, auch 3D + leicht erweiterbar durch objektorientierte Programmierung - hoher Berechnungsaufwand
Physikalisch fundiert: Gitter aus Widerständen (10)	Gitter aus Elementarwiderständen	Nicht veröffentlicht	?	?	hoch	ja	o	+ einheitliche Berechnungsformel für Leit- und Hauptblitz
Phänomenologisch, einfach (5)	Liniensegmente	Prozedural, Python	116	2	mittel	nein	-	- schwer erweiterbar durch prozedurale Programmierung
Phänomenologisch, Space Colonization (11)	Liniensegmente	Nicht veröffentlicht	?	?	mittel	ja	++	- Space Colonization wird nicht für Verästelungen genutzt, eigene Erweiterungen notwendig
Eigener Algorithmus	Liniensegmente	Objektorientiert, Java	210	3	gering	nein	+	+ leicht erweiterbar durch objektorientierte Programmierung