

# Integriertes Forschungsprojekt

zum Thema

## Analyse der Datenstrukturen mit dem Ziel der Erzeugung von Eingabedaten für das Graph Network-based Simulator Framework

an der Hochschule Darmstadt

Fachbereich Maschinenbau und Kunststofftechnik

vorgelegt von

**Daniel Roßnagel**

Goethestraße 23

75331 Engelsbrand

Matrikelnummer: 78841

Betreuer

Prof. Dr. Thomas Grönsfelder

Bearbeitungszeitraum

09. August bis 14. Oktober 2021

## Inhaltsverzeichnis

1	Abbildungsverzeichnis .....	2
2	Ausgangssituation .....	3
3	Zielsetzung.....	3
4	Anwendung nach Anleitung der Autoren.....	3
4.1	Verwendete Tools .....	4
4.2	Beispielhafte Anwendung .....	4
5	Analyse der Struktur der Eingabedaten .....	5
6	Erstellen eines eigenen Datensatzes.....	8
7	Zusammenfassung und Ausblick .....	10
8	Literaturverzeichnis.....	10

## 1 Abbildungsverzeichnis

Abbildung 1: Beispielhafte Ausgabe des GNS Frameworks zur Simulation von Wasser ( $t_{\text{sim}} = 0\text{s}$ ) .....	3
Abbildung 2: Beispielhafte Ausgabe des GNS Frameworks zur Simulation von Wasser ( $t_{\text{sim}} = 1\text{s}$ ) .....	3
Abbildung 3: Datenstruktur eines Datensatzes am Beispiel WaterDropSample .....	5
Abbildung 4: Übersicht der Partikeltypen, Quellcodeauszug aus render_rollout.py:47.....	5
Abbildung 5: Ausschnitt der Animation MultiHippo (Learning to simulate 2021).....	5
Abbildung 6: Gekürzter Auszug der Konsolenausgabe von read_tfrecord.ipynb, Context_Features ....	6
Abbildung 7: Gekürzter Auszug der Konsolenausgabe von read_tfrecord.ipynb, Data-Features .....	6
Abbildung 8: Positionsdatenstruktur der tfrecord-Datensätze am Beispiel test.tfrecord aus WaterDropSample.....	7
Abbildung 9: Positionsdatenstruktur der geladenen Datensätze am Beispiel test.tfrecord aus WaterDropSample.....	7
Abbildung 10: Datenstruktur des geladenen und bearbeiteten Datensatzes .....	7
Abbildung 11: Gekürzter Auszug der Konsolenausgabe von read_preprocessed_tfrecord.ipynb .....	8
Abbildung 12: Ausgabe des modifizierten Datensatzes ( $t_{\text{sim}} = 0\text{s}$ ) .....	9
Abbildung 13: Ausgabe des modifizierten Datensatzes ( $t_{\text{sim}} = 1\text{s}$ ) .....	9
Abbildung 14: Aufbau des particle_type-Arrays zum Erstellen eines eigenen Datensatzes.....	9
Abbildung 15: Ausgabe des modifizierten Datensatzes mit soliden Partikeln ( $t_{\text{sim}} = 0\text{s}$ ) .....	9

## 2 Ausgangssituation

Der Anlass für dieses Projektes ist das Graph Network-based Simulators Framework sowie das Paper von Sanchez-Gonzalez et al. Dies ist ein Framework für maschinelles Lernen, mit dem eine Vielzahl an physikalischen Anwendungen simuliert werden kann. Dazu gehört, wie Flüssigkeiten, starre Körper oder auch verformbare Körper in einer definierten Umgebung miteinander interagieren (Sanchez-Gonzalez et al. 2020). Im weiteren Text wird Graph Network-based Simulator mit GNS abgekürzt.

Speziell für die Simulation von Fluiden ist dies sehr interessant, da so mit einem vortrainierten Modell die Strömung von Flüssigkeiten oder Gasen in CAD-Bauteilen näherungsweise ermittelt werden kann, ohne ein rechenintensives Simulationstool wie Ansys CFX verwenden zu müssen (Ansys CFX 2021).

Wie dieses Framework jedoch strukturiert oder zu bedienen ist, um eigene Aufgaben damit bearbeiten zu können, ist an der Hochschule Darmstadt noch unklar. Um das Framework in Zukunft für eigene Anwendungen zu nutzen, soll dies im Umfang des IFPs erarbeitet werden.

## 3 Zielsetzung

Die Aufgabenstellung des Projektes besteht darin, sich mit der Struktur des Frameworks vertraut zu machen. Hierbei ist nicht der Prozess des maschinellen Lernens gemeint, sondern die Strukturen darum, also wie die Datensätze aufzubauen sind, um das GNS Framework zu nutzen und um damit eigene Modelle zu trainieren, aber auch schon vorhandene Modelle anzuwenden. Darüber hinaus geht es um die Handhabung des Frameworks, d. h. welche Befehle sind auszuführen, um Ergebnisse zu erzeugen oder um ein, wie oben angesprochen, eigenes Modell zu trainieren.

Die Ergebnisse und Informationen, die hierbei gesammelt werden, sollen daraufhin in diesem Paper dokumentiert werden. Abschließend wird das Vorgehen des Reverse-Engineering Prozesses erläutert, um so nachfolgenden Interessenten eine Guideline zu geben, wie sich diese selbst in das GNS Framework einarbeiten können.

## 4 Anwendung nach Anleitung der Autoren

Das GNS Framework simuliert die Interaktion von Partikeln untereinander in einer definierten Umgebung. In Abbildung 1 ist eine beispielhafte zweidimensionale Ausgabe zu sehen. Es handelt sich hier um einen Bildausschnitt einer animierten Darstellung.

Zu erkennen ist auf der linken Seite die „Ground truth“ Simulation. Abgebildet werden also die Positionen der einzelnen Partikel, welche durch eine Simulationssoftware anhand von physikalischen Modellen ermittelt wurden. Auf der rechten Seite hingegen ist die „Prediction“ Simulation. Dies ist die Vorhersage des im Vorfeld trainierten Modells. Rechts in Abbildung 2 ist dieselbe Animation zu einem späteren Zeitpunkt dargestellt. Die Diskrepanz zwischen der „Ground truth“ und „Prediction“ wird hier nicht weiter diskutiert, da dies nicht bestand des Projektes war.

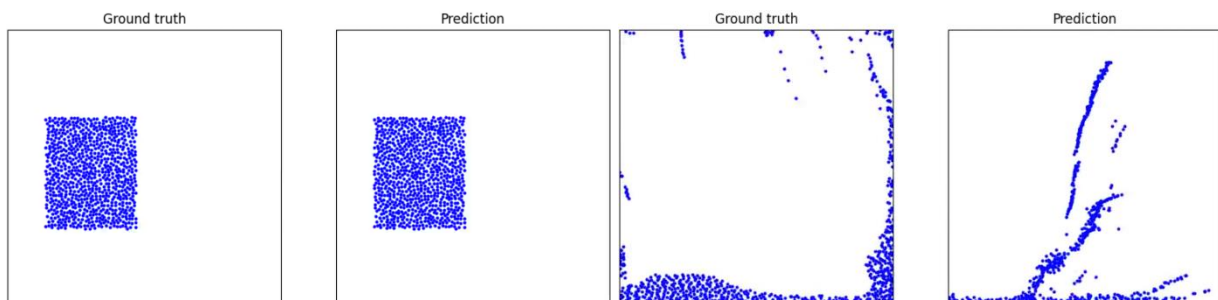


Abbildung 1: Beispielhafte Ausgabe des GNS Frameworks zur Simulation von Wasser ( $t_{sim} = 0s$ )

Abbildung 2: Beispielhafte Ausgabe des GNS Frameworks zur Simulation von Wasser ( $t_{sim} = 1s$ )

## 4.1 Verwendete Tools

Während des Projektes wurde das interaktive webbasierte Notebook Interface von JupyterLab verwendet. Dies wurde aus einer virtuellen Umgebung aus Anaconda heraus gestartet. So wurde unter Windows eine Umgebung erstellt, in der mit vielen verschiedenen Programmiersprachen entwickelt werden kann. Die Notebooks von JupyterLab erlauben es, Python Code selbst zu schreiben, auszuführen und auch zu teilen (The Jupyter Notebook 2021). Somit ist es für Interessenten möglich den für dieses Projekt verwendeten Code auszuführen.

## 4.2 Beispielhafte Anwendung

In diesem Abschnitt wird anhand eines Beispiels die ganzheitliche Anwendung des GNS Frameworks erläutert. Diese Befehle können entweder in einer zweiten Instanz der Anaconda Prompt ausgeführt werden oder in einem Konsolenfenster von JupyterLab.

1. Vor der ersten Anwendung empfiehlt es sich eine übersichtliche Ordnerstruktur für das Arbeiten mit den Datensätzen aufzubauen. Es wird hier jeweils ein Ordner für die Datensätze, die trainierten Modelle und die Vorhersagen angelegt.

```
mkdir datasets models rollouts
```

2. Die notwendigen Datensätze können entweder in [GitLab](#) heruntergeladen werden oder über die Anleitung die [hier](#) zu finden ist. Für einen schnellen Start empfiehlt sich der WaterDropSample Datensatz. Dieser muss im datasets Ordner abgelegt werden.
3. Nun kann das erste Modell trainiert werden:

```
python -m learning_to_simulate.train --data_path=./datasets/WaterDropSample --  
model_path=./models/WaterDropSample
```

4. Mit dem erfolgreich erstellen Modell können Trajektorien des Test-Datensatzes erzeugt werden:

```
python -m learning_to_simulate.train --data_path=./datasets/WaterDropSample --  
model_path=./models/WaterDropSample --output_path=./rollouts/WaterDropSample/ --  
mode="eval_rollout"
```

5. Zum Schluss können einzelne Animationen erzeugt werden:

```
python -m learning_to_simulate.render_rollout --  
rollout_path=./rollouts/WaterDropSample/rollout_test_0.pkl
```

Im weiteren Verlauf wird lediglich der Schritt des eval\_rollout näher analysiert, da hierbei mit einem bereits trainierten Modell und eigenen Datensätzen Vorhersagen getroffen werden können. Für diese Analyse wird der Aufbau der gegebenen Datensätze zurückverfolgt sowie das Preprocessing der Daten vor dem Schritt des Vorhersagens genauer betrachtet.

## 5 Analyse der Struktur der Eingabedaten

Für ein besseres Verständnis bezüglich der gegebenen Dateistruktur folgt, in Abbildung 3 am Beispiel WaterDropSample, eine Übersicht aller Dateien, die zu einem Datensatz gehören.

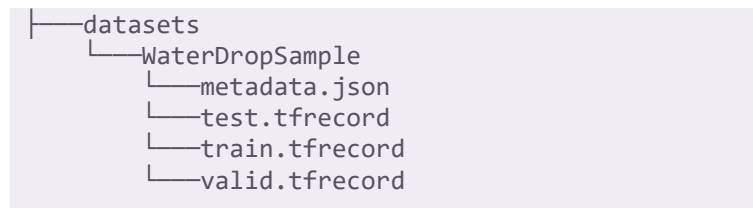


Abbildung 3: Datenstruktur eines Datensatzes am Beispiel WaterDropSample

Jeder Datensatz besteht aus vier Dateien, drei Dateien des tfrecord-Typs und eine vom Typ json. In dem Metadata-File sind allgemeine Informationen wie die Schrittzahl, die Begrenzungen der Animation, die Zeitschrittweite, die Standard- und Durchschnittsbeschleunigung und -geschwindigkeit zu finden. Die drei tfrecord-Dateien werden für das Training, die Validierung und das Testen eines Modells benötigt. Sie gehören zum open-source Framework Tensorflow und sind im Framework eigenen tfrecord-Format angelegt. Vor der Bearbeitung des Projektes ist der genaue Inhalt dieser drei Dateien nicht bekannt, da diese eine Bytes-Struktur aufweisen, welche mit einfachen Texteditoren nicht auslesbar ist (Build TensorFlow input pipelines 2021). Es ist jedoch zu erwarten, dass diese Dateien die Positionsinformationen aller Partikel zu den jeweiligen Zeitschritten enthalten. Ebenfalls sollten Sie weiteren Kontext zum Typ der einzelnen Partikel erhalten, da es hier fünf verschiedene Typen gibt (Abbildung 4). Diese Typen werden später in der Animation mit verschiedenen Farben dargestellt (Abbildung 5). Jeder Typ hat ein charakteristisches Verhalten, welches das Modell während des Trainingsprozesses lernt.

```
TYPE_TO_COLOR = {
    3: "black", # Boundary particles.
    0: "green", # Rigid solids.
    7: "magenta", # Goop.
    6: "gold", # Sand.
    5: "blue", # Water.
}
```

Abbildung 4: Übersicht der Partikeltypen, Quellcodeauszug aus `render_rollout.py:47`

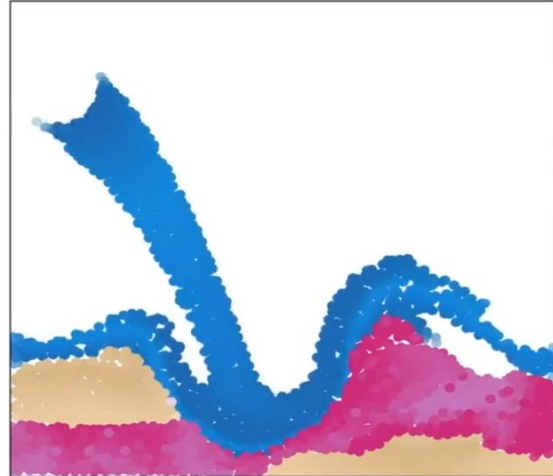


Abbildung 5: Ausschnitt der Animation MultiHippo (Learning to simulate 2021)

Für die Analyse der originalen unbearbeiteten Datenstruktur in den tfrecord-Datensätzen wurde das `read_tfrecord.ipynb` erstellt. Wird das Notebook ausgeführt ist zu erkennen, dass ein tfrecord-Datensatz aus mehreren Features besteht. Diese setzen sich zusammen aus Context (Abbildung 6) und Data (Abbildung 7). In den Context-Features werden, wie es der Name schon sagt, Kontextinformationen gespeichert und in den Data-Features sind die eigentlichen Daten enthalten. In diesem speziellen Fall setzen sich die Context-Features zusammen aus `particle_type`-Feature und `key`-Feature, wobei ersteres den Typ der einzelnen Partikel definiert. Der Hintergrund für die Angabe des Key-Features hingegen, wurde während der Bearbeitung des Projektes nicht aufgedeckt. Dies wird, nach dem bisherigen Verständnis, nirgends im Quelltext verwendet.

```
Context:  
particle_type: SparseTensorValue(indices=array([[0]], dtype=int64),  
values=array([b'\x05\x00\x00\x00\x00\x00\x00\x00\x05\x00\x00\x00\x00\x00\x00\x00  
\x05\x00\x00\x00\x00\x00\x00\x00\x05\x00\x00\x00\x00\x00\x00 ...  
\x05\x00\x00\x00\x00\x00\x00\x00']),  
dtype=object), dense_shape=array([1], dtype=int64))  
key: 1
```

In Abbildung 7 werden die Data-Features näher betrachtet. Diese bestehen aus dem position-Feature, welches in diesem Fall ein Array mit 1001 Feldern ist. Der Struktur der Einträge pro Feld zufolge, sind die Werte ebenfalls durch Serialisierung in eine binäre Form umgewandelt worden. Die Einträge eines Feldes wurden in Abbildung 7 gelb hervorgehoben. Es fällt auf, dass nicht das gesamte Array am Stück serialisiert wurde, sondern jedes Feld des Arrays für sich.

```
Data:
position: SparseTensorValue(indices=array([[ 0, 0],
      [ 1, 0],
      [ 2, 0],
      ...,
      [ 998, 0],
      [ 999, 0],
      [1000, 0]], dtype=int64),
values=array([b'\xafJ\''?\xe9a\xac>\xd8v\''?\xcFY\xa5>\x06"\''?\x98\xe6\x9c>\x06\x9
a$?i\x8b\xa4>9($?\xc0m\xa0>\xfdq?\x0f\xfa\xa8>\xab\x15&?Ch\x93>\xaf{'\''?\x0fa\x
8>\x14^&?\xb3g\xb2>\x0f?&?\x04\xe1\x87>. ... \n\x17?T\xda\xfo>n\x12\x1c?\x0b\x1e
b>Wk\x1a?k\x30>\xe6 &?\x8f\xf2\xe8>\xe6\xf4\x13?<\x8b\xf2>',
b'\xafJ\''?\xaaU\xac>\xd8v\''?\x93M\xa5>\x06"\''?]\xda\x9c>\x06\x9a$?* \x7f\xa4>9($?
\x87a\xa0>\xfdq?\xd3\xed\xa8>\xab\x15&?\x0b\\\x93>\xaf{'\''?\xd3T\xb8>\x14^&?s[\x
b2>\x0f?&?\xcb\xd4\x87>.\xb7%?\xeb\xb2\x8c>\\\xd0#?\xafA\x8f>vH%? ...
e\xe3=4Q\xda>,i\xea='],
dtype=object), dense_shape=array([1001, 1], dtype=int64))
```

Bei dem WaterDropSample Datensatz handelt es sich um ein Array mit 1001 Feldern, da sich dieser Datensatz aus 1000 Zeitschritten zusammensetzt. Hinzu kommt noch die Position der Partikel bei  $t=0[\text{sek}]$ , somit ergeben sich 1001 Felder.

In jedem der Felder des Arrays befinden sich die Positionsinformationen aller Partikel zu dem jeweiligen Zeitschritt. Die Struktur des Positions-Arrays ist somit `[Zeitschritte+1, Partikelanzahl, 2]`. Dass die letzte Dimension zwei beträgt, liegt daran, dass immer der x- und y-Wert jedes Partikels angegeben werden. Grafisch dargestellt ist diese Struktur in Abbildung 8.

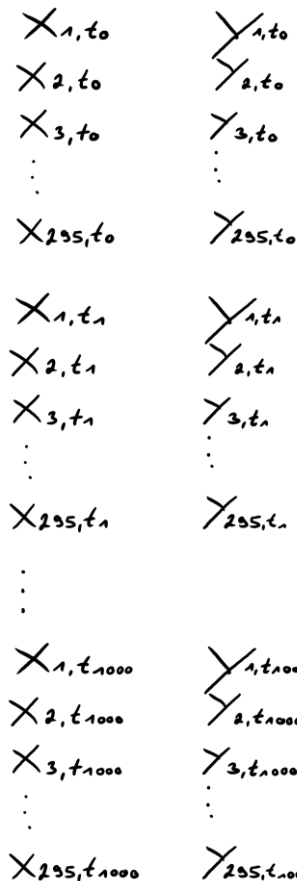


Abbildung 8: Positionsdatenstruktur der tfrecord-Datensätze am Beispiel test.tfrecord aus WaterDropSample

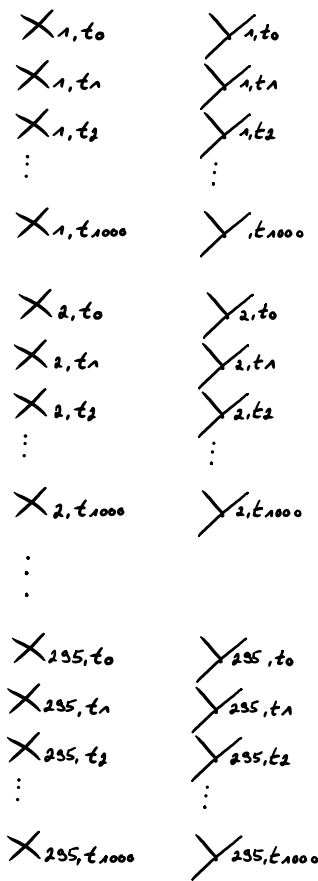


Abbildung 9: Positionsdatenstruktur der geladenen Datensätze am Beispiel test.tfrecord aus WaterDropSample

Zunächst wurde mit den Datenformat nach dem Preprocessing gearbeitet. Also nachdem die Datensätze geladen und bearbeitet wurden. Dieses sind wie in Abbildung 11 erkennbar gegliedert. Es ist zu erkennen, dass sich die Datenstruktur im Vergleich zur Struktur in den tfrecord-Files geändert hat. Es handelt sich hier um ein out\_dict, also ein Ausgabe-Dictionary, in anderen Worten ein Nachschlagewerk, und die target\_position, also die letzte Position der Partikel. Der Hintergrund für die Angabe vom target\_position ist wie beim Key-Feature zum jetzigen Standpunkt noch immer nicht geklärt. Ein Überblick der gesamten Datenstruktur verschafft Abbildung 10. Das out\_dict-Dictionary setzt sich zusammen aus mehreren tf.Tensors. Dies ist ein Tensorflow eigener Datentyp, der immer aus einem mehrdimensionalen Array besteht (TensorFlow 2021). Es ist hier jeweils die ID, die Form, der Datentyp des Inhalts und der Inhalt selbst angegeben. In diesem Datensatz befinden sich 295 Partikel mit 1000 Zeitschritten. Die Struktur des Positions-Arrays hat sich geändert. Sie lautet nun [Partikelanzahl, Zeitschritte, 2]. Es sind hierbei also jeweils alle Zeitschritte pro Partikel abgelegt und nicht wie vorher alle Zeitschritte und die Position der Partikel zu diesem. Eine Verdeutlichung schafft Abbildung 9.

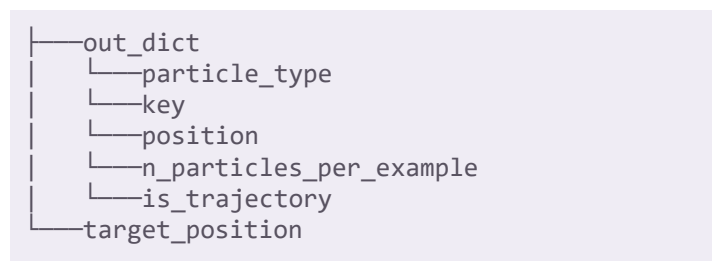


Abbildung 10: Datenstruktur des geladenen und bearbeiteten Datensatzes

```

out_dict: {'particle_type': <tf.Tensor: id=12143, shape=(295,), dtype=int64,
numpy= array([5, 5, 5, 5 ... 5, 5, 5, 5], dtype=int64)>, 'key': <tf.Tensor:
id=12141, shape=(), dtype=int64, numpy=0>, 'position': <tf.Tensor: id=12144,
shape=(295, 1000, 2), dtype=float32, numpy=
array([[0.65348333, 0.3366845 ],
       [0.65348333, 0.33659106],
       [0.65348333, 0.3364364 ],
       ...,
       [0.8152361 , 0.10710638],
       [0.8151884 , 0.10710894],
       [0.81514347, 0.10711136]],
       ...,
       [0.4243241 , 0.1144923 ],
       [0.42501888, 0.11448085],
       [0.42571098, 0.11446922]]], dtype=float32)>, 'n_particles_per_example':
<tf.Tensor: id=12142, shape=(1,), dtype=int32, numpy=array([295])>,
'is_trajectory': <tf.Tensor: id=12140, shape=(1,), dtype=bool, numpy=array([
True])>}]

target_position: [[0.8151003  0.10711416]
 [0.8009222  0.10607563]
 [0.83261764  0.1068546 ]
 [0.6630566  0.10649024]
 ...
 [0.5652377  0.09820223]
 [0.50735444  0.11089735]
 [0.42640078  0.11445841]]

```

Abbildung 11: Gekürzter Auszug der Konsolenausgabe von `read_preprocessed_tfrecord.ipynb`

Ein Datensatz besteht nicht ausschließlich aus einem Satz an Daten. Es ist möglich, mehrere Datensätze, in diesem Fall also mehrere Partikelverläufe zu verschiedenen Beispielen aneinander zu hängen und somit einen größeren Datensatz zu erzeugen. Der `WaterDropSample` Satz besteht beispielsweise aus zwei verschiedenen Szenarien. Das Erste besteht aus einem Partikelverlauf mit 295 Partikeln und das zweite Szenario besteht aus 803 Partikeln.

Mit diesem Wissen über die nun im Klartext erhaltenen Informationen zu den Datensätzen, kann ein eigener Datensatz erzeugt werden.

## 6 Erstellen eines eigenen Datensatzes

Durch die am Ende des Projektes gesammelten Informationen darüber, wie die Rohdaten im Datensatz gespeichert werden, sollte es möglich sein, eigene `tfrecord`-Files zu erzeugen.

Während des Projektes waren diese Informationen noch nicht vorhanden. Deswegen wurde die Variante gewählt, nach dem Preprocessing der Daten einen Datensatz mit derselben Struktur zu erzeugen wie die des originalen Datensatzes und diesen dann zu überschreiben.

Wie dies implementiert ist, kann im `create_Dataset.ipynb` Notebook nachgeschaut werden. Es wird für jeden Tensor im `out_dict`-Dictionary sowie für das `target_position`-Array aus Abbildung 10 ein alternatives Array erzeugt. Diese Arrays werden mit den dafür vorgesehenen Werten gefüllt. In Abbildung 14 ist beispielhaft das `particle_types`-Array dargestellt. Die einzelnen Arrays werden anschließend in Tensoren umgewandelt, welche daraufhin zusammengefügt werden, um einen Datensatz zu erzeugen.

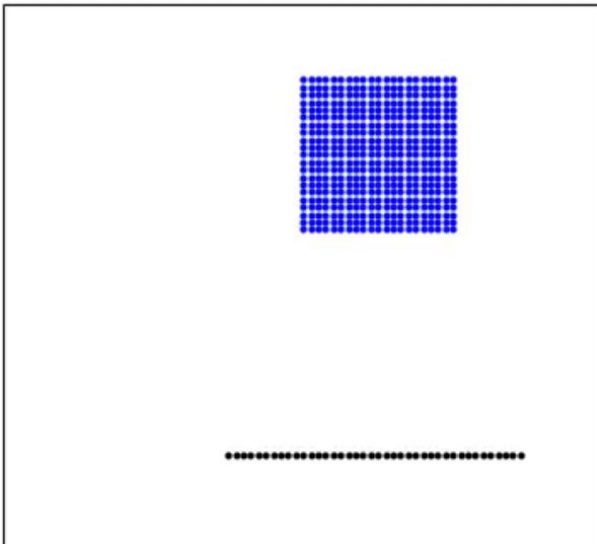


[illegible]

Abbildung 13: Ausgabe des modifizierten Datensatzes ( $t_{sim} = 1s$ )

```
particle_type: [3 3 3 3 3 3 3 3 3 3 3 3 3 3  
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3  
3 3 3 3] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
...  
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5]
```

Ground truth



The ground truth visualization consists of two distinct sets of points. The first set is a 10x10 grid of blue dots, forming a square shape in the upper right quadrant of the plot. The second set is a horizontal line of black dots, located in the lower left quadrant, consisting of 20 dots arranged in a single row.

Zusätzlich dazu muss in jedem Zeitschritt im `positions`-Array immer dieselbe Position jedes hinzugefügten Partikels angegeben werden. Hierbei gilt auch wieder, gleich wie beim `particle_type`-Array, am Anfang oder am Ende.

## 7 Zusammenfassung und Ausblick

Es wurde gezeigt, dass es funktioniert, eigene Daten auf ein bereits trainiertes Modell anzuwenden. Ebenso sollte es mit den nun vorhandenen Informationen über die Rohdaten eines Datensatzes möglich sein, selbst tfrecord-Files zu erzeugen und somit das GNS Framework in der regulären Art und Weise zu nutzen.

Der Aufbau der tfrecord-Datensätze muss sich hierbei folgendermaßen gestalten. In den Context-Features befinden sich die Informationen zum particle\_type der Partikel sowie ein Key-Feature. Das Key-Feature ist lediglich ein Element mit dem Wert eins. Das particle\_type-Feature ist als Array vom Typ int64 anzulegen und dann durch Serialisierung in ein Byte-Format zu bringen. Siehe hierfür Abbildung 6. Die Data-Features enthalten das position-Feature. Das Array der Positionsdaten ist in der Struktur nach Abbildung 8 anzulegen. Danach muss jedes Feld des Arrays serialisiert und somit in die Byte-Form gebracht werden. Jedes Feld enthält hierbei die Positionsdaten aller Partikel zu einem Zeitschritt. Wird nicht mit eigenen tfrecord-Files gearbeitet, sondern der geladene Datensatz überschrieben, muss ein Datensatz mit dem Format nach Abbildung 10 erzeugt werden. Hierbei gilt es zuerst einzelne Arrays zu erzeugen, welche dann in Tensoren umgewandelt und aneinandergefügt werden. Die Positionsdaten sind hierbei wiederum nach dem Beispiel aus Abbildung 9 aufzubauen.

Mit weiterer Einarbeitung wird es zudem auch möglich sein, eigene Arten der Ausgabe für die Ergebnisse herzustellen, um so noch speziellere Anwendungen präsentieren zu können. Ein Beispiel wäre hierfür nicht nur die Darstellung der Partikel als einzelne Punkte pro Zeitschritt, sondern eine Präsentation als Pfeile, ähnlich einem Vektorpfeil, der die Position und die Bewegungsrichtung anzeigt.

## 8 Literaturverzeichnis

Ansys CFX (2021). Online verfügbar unter <https://www.ansys.com/products/fluids/ansys-cfx>, zuletzt aktualisiert am 25.10.2021, zuletzt geprüft am 25.10.2021.

Build TensorFlow input pipelines (2021). Online verfügbar unter [https://www.tensorflow.org/guide/data#consuming\\_tfrecord\\_data](https://www.tensorflow.org/guide/data#consuming_tfrecord_data), zuletzt aktualisiert am 05.10.2021, zuletzt geprüft am 11.10.2021.

Ernst, Hartmut; Schmidt, Jochen; Beneken, Gerd (Hg.) (2020): Grundkurs Informatik: Grundlagen und Konzepte für die erfolgreiche IT-Praxis – Eine umfassende, praxisorientierte Einführung. Wiesbaden: Springer Fachmedien Wiesbaden.

Learning to simulate (2021). Online verfügbar unter [https://sites.google.com/view/learning-to-simulate/home#h.p\\_hjnaJ6k8y0wo](https://sites.google.com/view/learning-to-simulate/home#h.p_hjnaJ6k8y0wo), zuletzt aktualisiert am 26.10.2021, zuletzt geprüft am 26.10.2021.

Numpy community (2021): NumPy Reference Release 1.21.0. Online verfügbar unter <https://numpy.org/doc/1.21/numpy-ref.pdf>.

Sanchez-Gonzalez, Alvaro; Godwin, Jonathan; Pfaff, Tobias; Ying, Rex; Leskovec, Jure; Battaglia, Peter W. (2020): Learning to Simulate Complex Physics with Graph Networks. Online verfügbar unter <https://arxiv.org/pdf/2002.09405>.

TensorFlow (2021): TensorFlow Core v2.6.0. tf.Tensor. Online verfügbar unter [https://www.tensorflow.org/api\\_docs/python/tf/Tensor](https://www.tensorflow.org/api_docs/python/tf/Tensor), zuletzt aktualisiert am 29.09.2021, zuletzt geprüft am 11.10.2021.

The Jupyter Notebook (2021). Online verfügbar unter <https://jupyter-notebook.readthedocs.io/en/stable/notebook.html>, zuletzt aktualisiert am 03.09.2021, zuletzt geprüft am 11.10.2021.