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 Einführung in das GNS Framework 3
4.1 Installation der verwendeten Tools 4
4.2 Beispielhafte Anwendung 5
5 Analyse der Struktur der Eingabedaten 5
6 Erstellen eines eigenen Datensatzes 8
7 Ausblick 9
8 Literaturverzeichnis 10

1 Abbildungsverzeichnis

Abbildung 1: Beispielhafte Ausgabe des GNS Frameworks zur Simulation von Wasser (t_{sim} = 0s) 3
Abbildung 2: Beispielhafte Ausgabe des GNS Frameworks zur Simulation von Wasser (t_{sim} = 1s) 3
Abbildung 3: Ausgabe des modifizierten Datensatzes (t_{sim} = 0s) 8
Abbildung 4: Ausgabe des modifizierten Datensatzes (t_{sim} = 1s) 8
Abbildung 5: Ausgabe des modifizierten Datensatzes mit soliden Partikeln (t_{sim} = 0s) 9

2 Ausgangssituation

Der Anlass für dieses Projektes ist das Graph Network-based Simulators Framework 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 rechenintensive Simulationstools verwenden zu müssen.

Wie dieses Framework jedoch strukturiert ist oder wie es 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

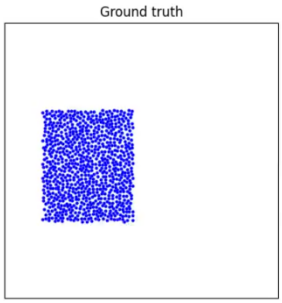
Die Aufgabenstellung des Projektes besteht darin, sich mit der Struktur des Frameworks vertraut zu machen. Hierbei ist nicht der Akt des maschinellen Lernens gemeint, sondern die Strukturen darum, also wie die Datensätze aufzubauen sind, um das GNS Framework verwenden zu können und um damit eigene Modelle trainieren, aber auch schon vorhandene Modelle nutzen zu können. Des Weiteren 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. Zusätzlich dazu 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 Einführung in das GNS Framework

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 den vorgegebenen Werten und den vorhergesagten Werten wird hier nicht weiter diskutiert, da dies nicht bestand des Projektes war.



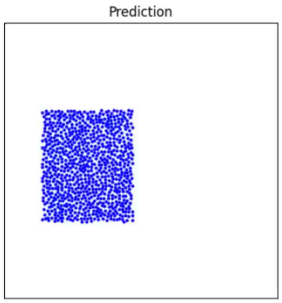
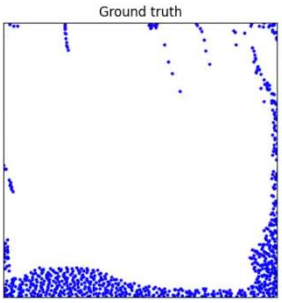


Abbildung 1: Beispielhafte Ausgabe des GNS Frameworks zur Simulation von Wasser (t_{sim} = 0s)



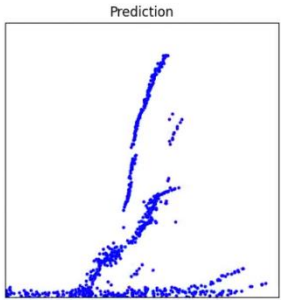


Abbildung 2: Beispielhafte Ausgabe des GNS Frameworks zur Simulation von Wasser (t_{sim} = 1s)

4.1 Installation der verwendeten Tools

Während des Projektes wurde das interaktive webbasierte Notebook Interface von JupyterLab verwendet. Dies wurde aus einer Anaconda Umgebung, kurz conda, 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).

Nachfolgend wird der Installationsprozess der während des IFPs durchgeführt wurde beschrieben.

Zur Installation von Anaconda den Installer auf dieser Webseite herunterladen und installieren. Anschließend in der Anaconda Prompt folgende Schritte ausführen, um eine eigene Umgebung zu erstellen in der JupyterLab installiert werden kann.

1. Erstellen einer Umgebung:

```
conda create --name myenv
```

Hinweis: myenv ist hierbei nur ein Platzhalter

2. Wenn conda nach einer Bestätigung fragt, mit *y* bestätigen:

```
proceed ([y]/n)?
```

3. Die conda Umgebung starten:

```
conda activate myenv
```

4. Nun kann JupyterLab heruntergeladen werden:

```
conda install jupyter-lab
```

5. Das JupyterLab Notebook wird gestartet mit:

```
jupyter-lab
```

Hinweis: Beim Starten des Notebooks können Parameter mit übergeben werden. Während des Projektes wurde das Notebook immer wie folgt gestartet. Dadurch kam es nicht zu Abstürzen, wenn viele Daten auf einmal in der Zellausgabe ausgegeben wurden.

```
jupyter-lab --NotebookApp.iopub_data_rate_limit=1.0e10 --
NotebookApp.iopub_msg_rate_limit=1.0e10
```

6. Anschließend muss das Repository von deepmind heruntergeladen werden. Hieraus kann dann das learning_to_simulate Projekt entnommen werden.
7. Jetzt müssen alle Abhängigkeiten zur Nutzung des GNS Frameworks installiert werden. Dafür muss das Arbeitsverzeichnis eine Ordnerstruktur über dem zuvor heruntergeladenen learning_to_simulate Ordner sein. Die Abhängigkeiten werden installiert mit:

```
pip install -r learning_to_simulate/requirements.txt
```

8. Am Ende werden die Ordner für die Datasets, Models und Rollouts erstellt:

```
mkdir datasets models rollouts
```

Nun sind alle Tools installiert und eingerichtet, damit selbst entwickelt werden kann.

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. 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 vorher angelegten datasets Ordner abgelegt werden.
2. Nun kann das erste Modell trainiert werden:

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

3. 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"
```

4. 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 in diesem Schritt 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

Das GNS Framework basiert wie viele andere Anwendungen zum maschinellen Lernen auf dem open-source Framework Tensorflow. Ein Format für die in Tensorflow verwendeten Datensätze ist das Framework eigenen tfrecord-Format. Hier werden die Daten in binärer Form gespeichert (Build TensorFlow input pipelines 2021).

Für die Analyse der originalen unbearbeiteten Datenstruktur in den tfrecord-Datensätzen wurde das read_tfrecord.ipynb erstellt. In der Ausgabe des Notebooks ist zu erkennen, dass ein tfrecord-Datensatz aus mehreren Features besteht. Diese setzen sich zusammen aus Context und Data. 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. Ein Partikel kann hierfür aus fünf verschiedenen Typen bestehen. Diese sind im Programmcode folgendermaßen definiert:

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

Diese Typen sind wichtig, um das Verhalten der einzelnen Sorten während des Trainings unterscheiden zu können und um das Plotten der Animation farblich besser zu gestalten.

Nachfolgend ist ein Auszug aus einem Druck der Context-Features zu sehen. Aufgrund der Syntax und des Datentyps des particle_type-Features kann zurückverfolgt werden, dass diese Informationen als int64 angelegt wurden, die vor dem Umwandeln in ein tfrecord-File zu einem byte-Array gecasted wurden. Dies ist zu erkennen, da ein durch Serialisierung in eine binäre Form gebrachter Wert stets mit `b'` anfängt (Numpy community 2021, S. 29–30). Darüber hinaus besteht ein int64 aus 64 Bits, es enthält somit acht Bytes (Ernst et al. 2020, S. 14). Demnach ist hier ein Array mit, in diesem Fall, nur Fünfen abgespeichert, wobei jede der Fünfen für ein Partikel steht. Die restlichen sieben Bytes jedes Wertes sind mit einer Null belegt.

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

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 Code verwendet.

Nun werden die Data-Features näher betrachtet. Diese bestehen lediglich aus dem position-Feature, welches ein Array mit 1001 Feldern ist. Der unleserliche Aufbau der einzelnen Einträge pro Feld mit einem `b'` am Anfang zeigt, dass die Werte wieder durch Serialisierung in eine binäre Form umgewandelt wurden.

```
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\xb
8>\x14^&?\xb3g\xb2>\x0f?&?\x04\xe1\x87>. ... \n\x17?T\xda\xf0>n\x12\x1c?\x0b\x1e
b>Wk\x1a?k\xf3O>\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 dieses Beispiel aus 1000 Zeitschritten zusammensetzt. Hinzu kommt noch die Position der Partikel bei t=0[sek], somit also 1001 Felder. Diese Information kann aus dem Metadata-File jedes Datensatzes entnommen werden. Hier heißt es demnach `"sequence_length": 1000`. Im Metadata-File sind zusätzliche Informationen wie die Begrenzungen der Animation, die Zeitschrittweite, die Standard- und Durchschnittsbeschleunigung und -geschwindigkeit zu finden.
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.

Diese detaillierte Struktur der rohen Datensätze wurde jedoch erst zum Ende des Projektes hin ermittelt, da das korrekte Auslesen der Daten zu Beginn des Projektes, mangels Wissens, noch nicht funktionierte. Deswegen wurde mit einem Datenformat nach dem Preprocessing gearbeitet. Dieses gliedert sich wie folgt:

```
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]]
```

Es ist zu erkennen, dass sich die Datenstruktur merklich verändert hat. Gehen wir hier auch wieder der Reihe nach vor. Zuerst sehen wir, dass der Aufbau der Features grundlegend anders ist. Hier handelt es sich nun nicht mehr um ein Context-Feature und ein Data-Feature, sondern um ein out_dict, also ein Ausgabe-Dictionary, und die target_position, also die letzte Position der Partikel. Der eigentliche Hintergrund für die Angabe vom target_position ist wie beim Key-Feature auch zum jetzigen Standpunkt noch immer nicht geklärt. Beim Betrachten des out_dict kann erkannt werden, dass es sich um ein Dictionary aus mehreren tf.Tensors handelt. 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 jedoch 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.

Ein Datensatz besteht nicht ausschließlich aus einem Satz an Daten. Es ist möglich, mehrere Datensätze, in diesem Fall also mehrere Patikelverläufe 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, können wir einen Eigenen Datensatz erzeugen.

6 Erstellen eines eigenen Datensatzes

Durch die am Ende des Projektes gesammelten Informationen darüber, wie die Rohdaten im Datensatz gespeichert werden, sollte es auch 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.
Im Notebook create_Dataset.ipynb ist zu sehen, wie dies während des Reverse-Engineerings umgesetzt wurde. Dieser Code wurde letzten Endes im originalen Code als Funktion eingefügt, um den vorhandenen Datensatz mit dem eigens erstellten zu überschreiben.
Als Beispiel ist in Abbildung 3 der modifizierte Datensatz zu Beginn der Simulation zu sehen. Die Partikel sind hier in immer gleichen Abständen vorhanden. Bei den originalen Datensätzen sind diese natürlicher verteilt. In der rechten Abbildung, Abbildung 4, ist die gleiche Animation zu einem späteren Zeitpunkt zu sehen. Auffällig ist hier, dass die Ground truth Partikel noch immer am selben Ort sind. Dies hat den Grund, dass für den einfachen Zweck der Demonstration die Partikel statisch über alle Zeitschritte hinweg am selben Punkt platziert wurden.

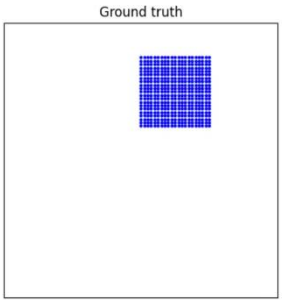
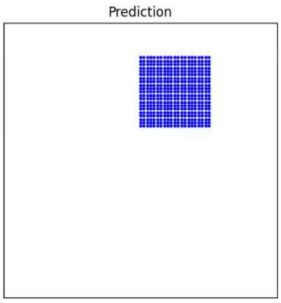
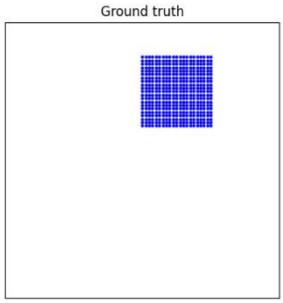


Abbildung 3: Ausgabe des modifizierten Datensatzes (t_{sim} = 0s)





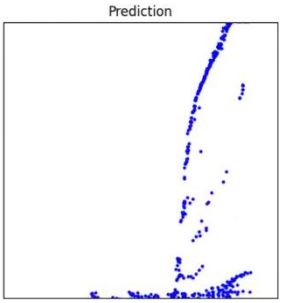
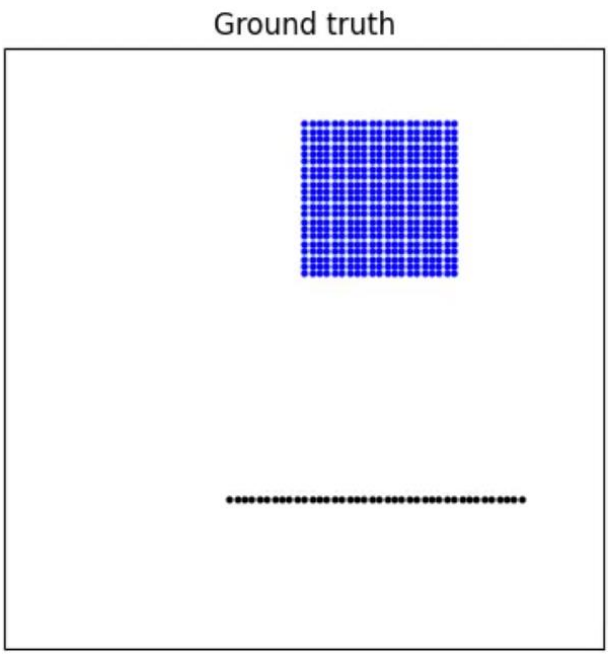


Abbildung 4: Ausgabe des modifizierten Datensatzes (t_{sim} = 1s)

Im Notebook create_Dataset.ipynb besteht ebenfalls die Möglichkeit, einen Datensatz mit verschiedenen Partikeltypen zu erzeugen. In dem dortigen Beispiel werden zusätzlich zu den Wasserpartikeln noch solide Hindernisse hinzugefügt (Abbildung 5). Um dies umzusetzen, muss das particle_type-Array und das position-Array verändert werden. Es muss hierfür zusätzlich für jeden Zeitschritt und jedes der soliden Partikel entweder am Anfang oder am Ende des particle_type-Array die Zahl für das Partikel angegeben werden. Dies sieht wie folgt aus:



```
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 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 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 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]
```

Abbildung 5: Ausgabe des modifizierten Datensatzes mit soliden Partikeln (t_{sim} = 0s)

Zusätzlich dazu muss in jedem Zeitschritt im position-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 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. Mit weiterer Einarbeitung wird es zudem 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 Präsentation der Partikel als einzelne Punkte pro Zeitschritt, sondern eine Darstellung als Pfeile, ähnlich einem Vektorpfeil, der die Position und die Bewegungsrichtung anzeigt.

8 Literaturverzeichnis

Build TensorFlow input pipelines (2021). Online verfügbar unter 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.

Numpy community (2021): NumPy Reference Release 1.21.0.

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, 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.