

GPU による人流シミュレーションの高速化

2311167 二宮 豪 成見研究室

1 はじめに

駅やイベント会場の混雑を検討するとき、歩行者の移動を計算する人流シミュレーションが使われる。避難経路や出入口の配置を複数の条件で比べたり、リアルタイムに計算するには、1 回の計算を短くすることが重要である。

本研究では、歩行者シミュレーション用の公開ライブラリ PySocialForce[1] を対象とする。

PySocialForce は社会力モデルに基づき、歩行者に働く力から位置と速度を繰り返し更新する。歩行者集団を避けるロボットの学習 [2] や、人混みを移動するロボットの研究基盤 [3] にも利用されている。しかし大人数シミュレーションにおいては計算が遅いため、本研究では GPU を用いて高速化する。

2 関連研究

Kemloh Wagoum[4] は、別の歩行者モデルに対して linked-cell 法を説明し、近傍探索の計算量を論じている。

本研究では、オープンソースである PySocialForce を GPU 化し、更に高速なアルゴリズムを適用する。

3 PySocialForce での力の計算方法

PySocialForce では、目的地へ向かう力、人を避ける力、壁を避ける力を計算する。歩行者数を N 、壁を表す点の数を M とすると、元の実装では各力の計算量はそれぞれ概ね $O(N)$, $O(N^2)$, $O(NM)$ である。特に人同士の力は、歩行者が増えると調べる相手の数が急増する。本研究では歩行者のグループ行動は扱わない。

高速化は順に 3 段階で進める。まず 3 種類の力の計算を歩行者ごとに並列化して GPU 上で実行する。次に、近くの歩行者だけを候補として取り出す近傍探索を導入し、人同士の計算を減らす。壁についても同様に、近い壁の点を先に探す方法を検討する。

4 GPU 化

4.1 GPU 化の方針

3 種類の力を歩行者単位の GPU カーネルで並列計算し、GPU 上で足し合わせる版を作成した。各更新では現在の位置と速度を GPU へ送り、計算後の合力を CPU へ戻す。位置・速度の更新と歩行履歴の保存は元の CPU 実装を使う。したがって、GPU 化したのは力の計算部分であり、シミュレーションの全処理を GPU に移したわけではない。この段階では近傍探索や距離による人同士の計算の打切りは導入していない。

GPU 版では、歩行者ごとに 1 つのスレッドを割り当てる。人を避ける力は各歩行者が他の歩行者を、壁を避ける力は壁点を調べるため、GPU 化だけでは調べる組合せの数は減っていない。

4.2 計算速度

30 m×20 m の領域の左側 20 m×20 m に 300 人を配置し、幅 4 m の出入口へ向かわせるシミュレーションを行った。時間刻みを 0.4 s として 300 回更新し、120 s 分の歩行を計算した。グループ行動は含めていない。CPU は Intel Core i3-14100F、GPU は NVIDIA GeForce RTX 4070 を用いた。

計算時間は、元の CPU 版と 3 種類の力を GPU 化した版をそれぞれ別の Python プロセスから 10 回起動した中央値で比較した。表 1 の時間には、プロセス起動、ライブラリの読み込み、シミュレータ生成、300 回の更新および保存した履歴の取得を含む。GPU の事前実行は行っていないが、ディスク上の既存 JIT キャッシュは残っている。この条件で GPU 版の所要時間は CPU 版の約 5.3 分の 1 となった。

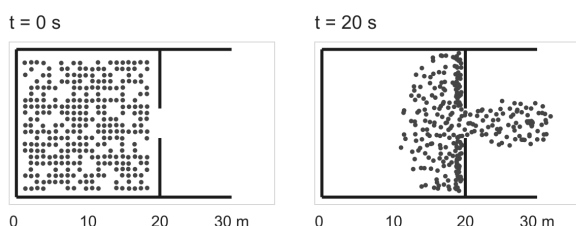


図 1 PySocialForce の CPU 版で計算した 300 人の歩行例（開始時と 20 秒後）



図 2 力の計算だけを GPU 化した処理の流れ

表 1 300 人・300 回更新の計算時間（各 10 回の中央値）

実装	計算時間	CPU 版に対する速度比
CPU 版	18.522 s	1.00 倍
GPU 版	3.485 s	5.31 倍

4.3 計算精度

歩行結果は、初期配置を 5 種類変え、各配置で同じ初期状態から CPU 版と GPU 版を実行して比較した。表 2 は 5 配置のシミュレーション結果の中央値である。t₃₀, t₆₀, t₉₀ はそれぞれ 30%, 60%, 90%の歩行者が出入口を通過するまでのシミュレーション内の時間であり、計算時間ではない。3 指標の中央値の相対差は最大約 2.7%だった。

一方、300 回更新後の歩行者ごとの位置の誤差は配置によって 4.36–4.64m であり、個々の軌跡は一致しない。また、GPU 版の 5 配置中 1 配置では、1 人が出入口以外の壁を横切ったと判定された。

CPU 版と GPU 版では計算順序や数値精度も異なるが、結果の差の原因はまだ特定できていない。

5 近傍探索による高速化の方針

元の人同士の力は、対象の歩行者について他の全員を計算する。近い人だけを計算対象にする場合には、全員との距離を毎回探索するのはコストがかかる。そこで、空間を格子に分けて歩行者を登録し、対象の歩行者がいる格子と周囲の格子から候補を取り出す linked-cell 法を検討する。候補に対して最後に距離を調べ、設定した範囲内の相手との力だけを計算する。文献 [4] では、別の歩行者モデルではあるが格子型の探索法として扱われている。

6 今後の評価方法

PySocialForce の人同士の力は遠方で直ちにゼロになる式ではない。したがって、一定の距離以上の人からの力を計算しない距離の打切りは計算結果も変えてしまう。このため人同士の計算については、現在の GPU 版を基準とし、(1) 全員との距離を調べて指定距離内だけ力を計算する版、(2) 同じ指定距離で linked-cell 法により候補を絞る版を作る。基準

と (1) の差から距離の打切りによる計算量と結果の変化を確認し、(1) と (2) の差から候補探索の変更による効果を確認する。

壁については、元的全壁点との距離判定と、静的な壁点を格子に登録して近傍候補を取得する方法を比べる。候補を取り出した後にも元の距離条件を適用し、壁を避ける力の計算結果を照合する。壁点の登録時間を含むシミュレーション全体の時間も測り、格子を作る費用を差し引いても利点があるかを調べる。

歩行結果は、同じ初期配置を用いて、t₃₀, t₆₀, t₉₀ だけでなく、時間ごとの出口通過人数、個々の位置の差を記録する。測定では、人数、密度、出口幅、壁の配置を変えながら結果を測る。

7 おわりに

現時点では、3 種類の力の計算を GPU に移し、300 人・300 回更新の一条件で所要時間が 18.522s から 3.485s に短くなった。今後は linked-cell 法による人の近傍探索と壁点の近傍探索を実装し、探索の準備時間を含む計算時間と歩行結果の両方を比較する。

参考文献

[1] Y. Gao, “PySocialForce,” <https://github.com/yuxiang-gao/PySocialForce> (accessed 2026-09-29).

[2] A. Katyal et al., “Learning a Group-Aware Policy for Robot Navigation,” *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 11328–11335, 2022.

[3] L. Kästner et al., “Demonstrating Arena 3.0: Advancing Social Navigation in Collaborative and Highly Dynamic Environments,” *Proc. of Robotics: Science and Systems*, Delft, Netherlands, 2024, doi:10.15607/RSS.2024.XX.074.

[4] A. U. Kemloh Wagoum, “Route Choice Modelling and Runtime Optimisation for Simulation of Building Evacuation,” doctoral dissertation, Universität Wuppertal, 2012; Jülich Supercomputing Centre, Forschungszentrum Jülich, IAS Series 17, 2013.

表 2 出入口を通過するまでの時間（5 配置の中央値, 単位：s）

実装	t ₃₀	t ₆₀	t ₉₀
CPU 版	21.6	44.8	84.4
GPU 版	21.6	46.0	85.2

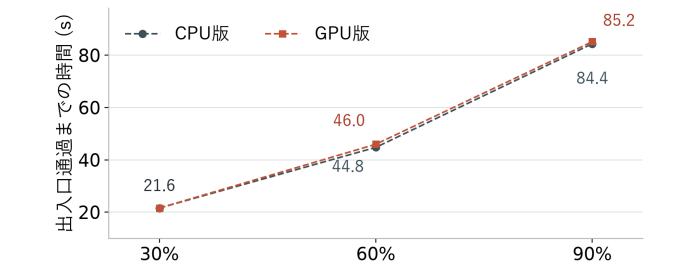


図 3 30・60・90%の歩行者が出入口を通過するまでの時間（5 配置の中央値）

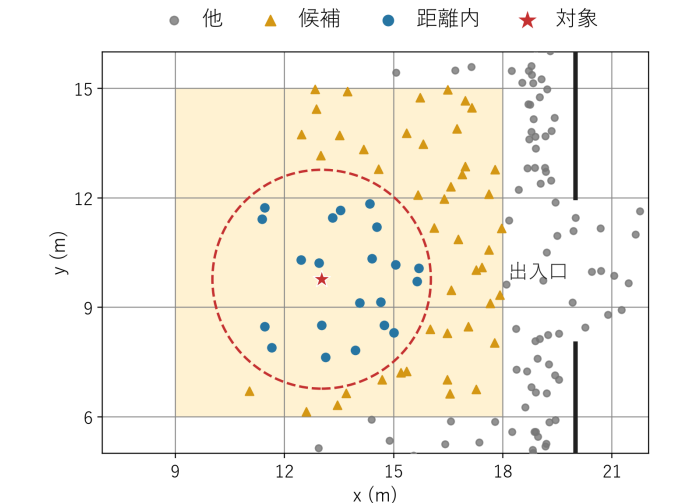


図 4 CPU 版 20 秒後の位置データで試した格子候補抽出