LEMONS: An open-source platform to generate non-circular, anthropometry-based pedestrian shapes and simulate their mechanical interactions in two dimensions

Oscar Dufour[1⋆], Maxime Stapelle[1‡], Alexandre Nicolas[1†]

1 Université Claude Bernard Lyon 1, Institut Lumière Matière, CNRS, UMR 5306, 69100, Villeurbanne, France

⋆ oscar.dufour@univ-lyon1.fr , ‡ maxime.stapelle@univ-lyon1.fr , † alexandre.nicolas@cnrs.fr

Abstract

To model dense crowds, the usual recourse to oversimplified (circular) pedestrian shapes and contact forces shows limitations. To help modellers overcome these limitations, we propose an open-source numerical tool. It consists of an online platform with a user-friendly graphical interface to generate 2D and 3D pedestrian crowds based on anthropometric data and a C++ library that computes mechanical contacts with other agents and with obstacles, and evolves the crowd's configuration. Both can readily be called from Python scripts and leave free reins to the user for the decisional layer of the model, i.e., the choice of the desired velocities.

Received Date
Accepted Date
Published Date

Contents

1 Introduction **2**
1.1 Motivations 2
1.2 How to read this document 4

2 Theory & Methods **5**
2.1 From the individual pedestrian's shape to the generation of a synthetic crowd 5
2.2 Mechanical interactions 6

3 The Codebase **8**
3.1 XML crowd configuration classes 8
3.2 Mechanical layer 10
3.3 Python classes 10

4 Discussion **12**
4.1 Relevance of the use of 2D projections of standing pedestrians 12
4.2 Mechanical tests 12
4.3 Practical case study 14
4.4 Extension to arbitrary shapes 17

5 Conclusion **17**

A Equation of motion 20
A.1 Mechanical interactions 20
A.1.1 Forces acting on the pedestrian centre of mass 20
A.1.2 Torque for rotation of a pedestrian 22
A.2 Moment of inertia calculation 22
A.3 Mechanical equations summary 22
B Mechanical layer: agent shortlisting 24
C Configuration files example 25
D Packing algorithm within the streamlit app 27
E Writing style 29
References 29

1 Introduction

1.1 Motivations

From an external physical standpoint, pedestrians are just mechanical bodies obeying Newton's equations of motion: their motion is constrained by physical interactions with the environment. They experience physical forces such as gravity and ground repulsion, which prevent sinking. Yet, they differ from inert objects in that they move autonomously without requiring external impetus. This reveals two intrinsically coupled levels of pedestrian dynamics: the mechanical level and the decision-making level. Crowd literature reflects this duality. Some studies focus on mechanical aspects (essential in high-density scenarios) [1,2] but most often relying on idealised interaction forces and simplified circular shape that fail to replicate mechanical interactions faithfully; others examine decision-making (especially relevant in low-density contexts) [3,4], while yet others [5] address the coupling of both levels, crucial in intermediate-density situations where individuals navigate to avoid collisions, but may nonetheless experience physical contact.

Most existing crowd dynamics models [2,5,7] represent pedestrians as disks. However, when using the bideltoid breadth (defined in Fig. 2) as the disk diameter, a tightly packed random arrangement of a realistic population (shown in Fig. 1) only achieves densities of about $4\,\mathrm{ped/m^2}$. This falls far short of empirically observed peak densities, which sometimes exceed $8\,\mathrm{ped/m^2}$ in real-world scenarios [8–10]. A potential solution to reconcile this density discrepancy would be to reduce disk diameters to the chest depth (instead of the bideldtoid breadth). However, this adjustment introduces critical flaws. First, it preserves the unrealistic circular body geometry, which fails to reflect human morphology and limits the number of simultaneous physical contacts by pedestrian to six, at most. In contrast, in controlled dense crowd scenarios [11,12], single individuals experiencing simultaneous contact with eight distinct others were observed. Second, narrower disks would artificially lift constraints on unidirectional flow in narrow corridors and overestimate the associated flow rate. Therefore, we turn to an alternative solution, using elongated shapes to represent the mechanical shape of a pedestrian.

The use of anisotropic shapes in the granular materials literature is well-established. Discrete element simulations have employed diverse geometries to describe solid dynamics: el-

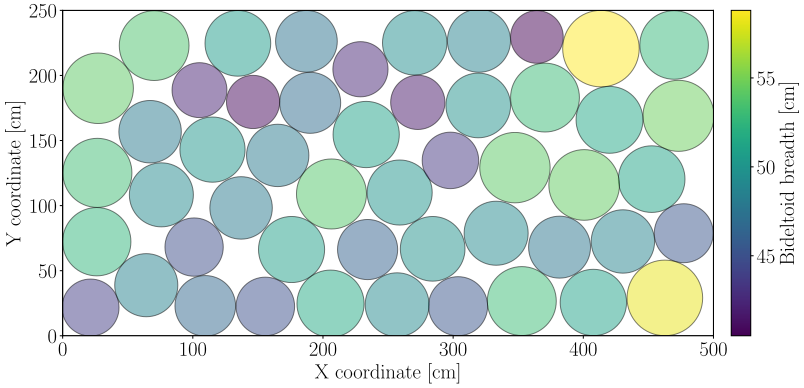

Figure 1: Tightly packed random pedestrian disk arrangement, reaching a density of 4 ped/m^2. The disk diameters are sampled from the empirical bideltoid breadth distribution of a US population subset (ANSURII database, [6]), with mean 49 cm and standard deviation 4 cm. Algorithm details: App. D.

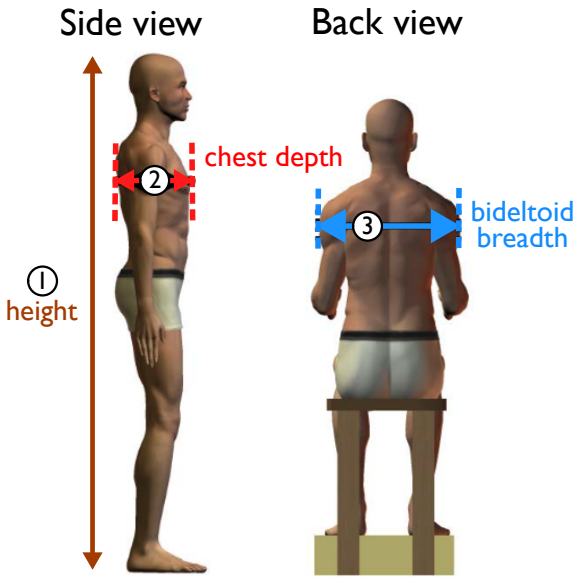

Figure 2: Illustration of anthropometric measurements – including height, chest depth and bideltoid breadth – adapted from [6].

lipses [13], polygons [14], polar-form polygons [15], and disk assemblies [16] represent key examples, while [17] provides a comprehensive review. Despite extensive research on granular dynamics, pedestrian models rarely integrate non-circular shapes within mechanical frameworks. Still, a number of exceptions are noteworthy. Elliptical volume exclusion was incorporated into generalised centrifugal force models, prioritising inertial forces over traditional damped-spring mechanics for pedestrian contact [18]. For models relying on the concept of velocity obstacles, the original circular agents' shapes were gradually extended to ellipsoids (EORCA), or polygonal approximations thereof for computational efficiency [19, 20], and to arbitrary shapes approximated by stitching rounded trapezoids centered on the medial axis of the shape [21]. These ellipsoidal or arbitrarily-shaped representations govern the choice of an optimal velocity that ideally enables collision-free navigation (decisional purpose), alien to any consideration of mechanical interactions. If one focuses on models including short-range and/or contact interactions, Langston et al. represented pedestrians with three overlapping circles in a discrete-element simulation [22], while spheropolygons [23] or spherocylin-

ders [24, 25] were later introduced in force-based simulations incorporating self-propulsion forces as well as granular material interactions governed by Newtonian mechanics, notably to model competitive egress scenarios. Recently, the human torso has been modelled as a capsule in a flow governed by position-based dynamics, supplemented with short-range interactions [26].

Nevertheless, albeit anisotropic, these shapes face significant limitations: they are more or less arbitrarily defined and lack a quantitative medical or anthropometric basis. Consequently, the generated crowds lack representative heterogeneity, which is crucial for accurately replicating density and collision statistics. These rigid structures also resist extension to new contact models involving deformation or relative motion between the centres of mass of the body segments. This article addresses these limitations by introducing a tool that generates realistic crowds from anthropometric data, simulates mechanical interactions, and allows user-defined decisional layers. It therefore removes the technical barriers of modelling elongated crowd shapes, allowing the community to focus on decision-making and its interaction with mechanics. This tool also opens the door to exploring essential questions about introducing complexity into modelling, such as whether to introduce a third dimension or incorporate heterogeneity in agent types, like strollers or individuals carrying bags.

In addition to serving researchers in the field, this tool is designed for crowd modellers at all levels, starting from beginners, in their efforts to assist, e.g. public authorities and businesses; it provides them with the possibility to achieve more realistic simulations of dense (and possibly heterogeneous) crowds in a simple way. In this particular regard, existing simulation software, such as Iventis [27] and Vadere [28], fail to reflect the latest advances; our solution brings crowd simulation into the present.

Finally, the proposed tool, dubbed LEMONS, may also be of pedagogical interest. It can easily be integrated into classroom settings, enabling teachers and science communicators to simulate agents with minimal effort. It has the potential to spark interest in the physical sciences, particularly in the study of complex systems and active matter.

1.2 How to read this document

This document exposes the theoretical foundations of the LEMONS software tool and provides an overview of the code structure. A minimal usage example, detailed usage tutorials (along with Jupyter Notebooks), and a comprehensive API documentation of the classes and functions in LEMONS can be found online [29]. Great care has been taken in the development of the codebase to make it user-friendly, easy to expand, and maintainable, as detailed in App. E.

This article is structured as follows. We begin by outlining the theoretical foundations of the project, introducing a novel mechanical shape for pedestrians, describing the generation of realistic crowds based on anthropometric data. Sec. 2.2 also details the specification of mechanical interactions between agents' shapes and with any walls present in the environment. The document then provides an overview of the code structure in Sec. 3. Finally, in Sec. 4, we present an in-depth discussion of our model, outline the tests conducted to validate its implementation, and propose potential directions for future improvements. We also provide detailed instructions on how to run a crowd evacuation simulation in this section. Supplemental videos of the tests and of the practical case studied are also provided [30].

2 Theory & Methods

2.1 From the individual pedestrian's shape to the generation of a synthetic crowd

For realistic pedestrian shapes, we relied on medical data, specifically, cross-sectional images from two cryopreserved middle-aged cadavers (a male at 1 mm intervals and a female at 0.33 mm intervals) provided by the Visible Human Project [31]. Since simulations in 2D prevail in the field of pedestrian dynamics, we need to project the 3D shape onto a suitable effective 2D shape. To this end, we selected the cross-section at torso height, an example of which is shown in Fig. 3 for the male specimen; this choice is notably justified by the fact fatalities during crowd crushes often result from asphyxia and severe compression of the rib cage and lungs. We approximated the torso slice with a set of five partly overlapping disks: two for the shoulders, two for the pectoral muscles, and one for the back, as illustrated in Fig. 7. Disks were chosen over polygons because defining and computing mechanical contact between disks is much simpler and more computationally efficient [17].

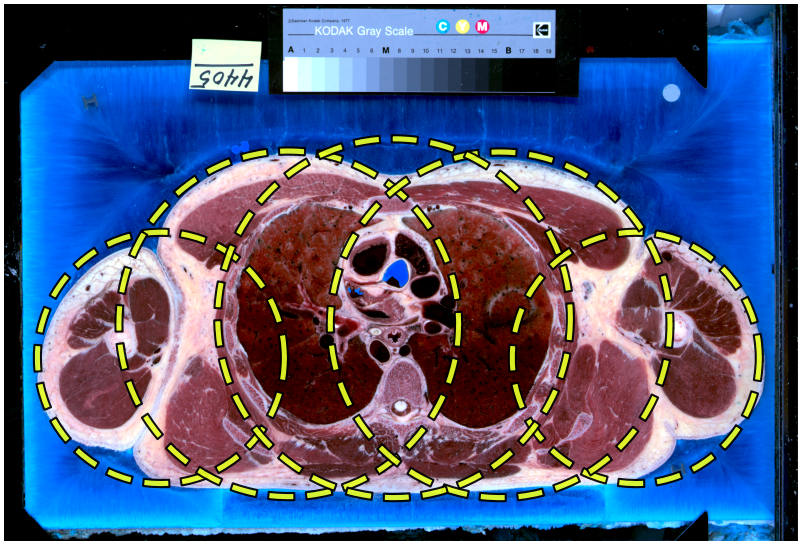

Figure 3: Torso section of a cryopreserved man, slice number 4405, from the [31] database, covered with five disks. The 'Kodak Q-13 Gray Scale' ruler measures 20.3 cm by 2.5 cm.

To extend the fitting method to a whole population, we utilised anthropometric data from the ANSURII database [6], which comprises 93 body measurements from 6,000 US Army personnel (4,082 men and 1,918 women). In the Anthropometry tab of our online app, these data are easily accessible, viewable, and downloadable. Note, however, that this sample is not fully representative of the US civilian population; in particular, among other selection biases, men are over-represented, whereas women form the majority of the US population, according to the NHANES database [32] [1] (which can be partly explained by the higher life expectancy of women in the US population). To generate a crowd that reflects the anthropometric diversity of ANSURII starting from the foregoing 2D projection made of five disks, we translate the centres of the disks with a homothety centred at the pedestrian's centre of mass and scale their radii to match empirical chest depth and bideltoid breadth measurements (defined in Fig. 2). These geometric operations do not perfectly preserve the initial shape, but they achieve a real-

[1] NHANES provides only limited measurements and lacks key metrics such as bideltoid breadth and chest depth and therefore cannot be used in our software.

istic approximation. Compared to the maximal possible density around 4 ped/m^2 for circular agents (see Fig. 1), crowds generated with these methods can reach a density of 7.2 ped/m^2 (see Fig. 4), much closer to empirical measurements in very dense situations [8–10].

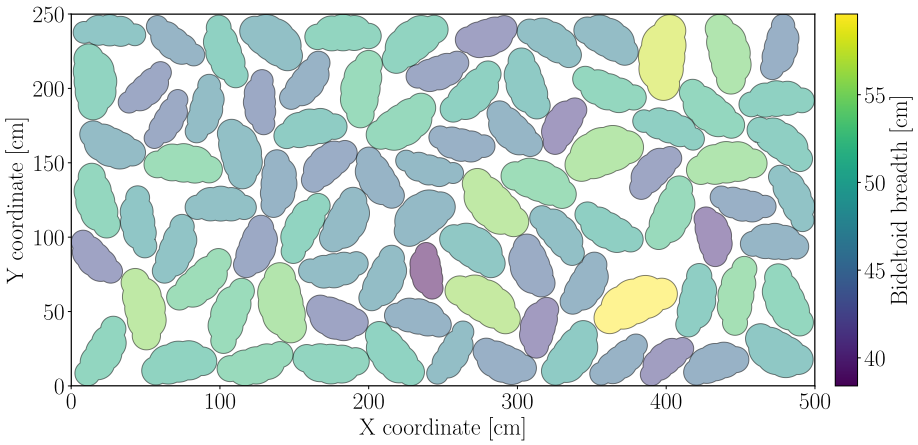

Figure 4: Tight random packing of pedestrians without preferred orientation using an arrangement of five disks, reaching a density of 7.2 ped/m^2. Both the sample from the *ANSURII* database [6] and our model database have a mean bideltoid breadth of 49 cm and a mean chest depth of 25 cm.

2.2 Mechanical interactions

Studying the mechanical interactions between pedestrians presents inherent complexity, arising from factors such as three-dimensional contact geometry, protective hand movements during contacts or falls, and non-static multi-point contacts [12,33]. Even though bio-mechanical studies have characterized stress-strain relationships in fresh, non-rigid cadavers subjected to dynamic loading — both frontal [34, 35] and lateral [36] — and have measured contact forces among pedestrians under varying degrees of crowding, in both static and dynamic conditions [37], the fundamental nature of live pedestrian-to-pedestrian contact remains poorly understood. This is especially true during complex, multi-body collisions, where active interventions, such as the use of hands, may significantly influence the interaction dynamics. To make the problem tractable, we deliberately simplify these interactions by relying on granular material interactions between the disks that constitute each agent's shape. Specifically, we model the interaction in the simplest way we find appropriate, using a single-damped spring as illustrated in Fig. 5. The stick-and-slip process at a dynamic contact follows Coulomb's law. Another force is introduced to encompass the effective backwards friction with the ground over a step cycle, controlled by the deformation of the body. Technical details are given in App. A.1, and a comprehensive overview of notations, definitions, and mathematical expressions can be found in App. A.3.

Finally, the deliberate forward motion is subsumed into a propulsion force $\mathbf{F}_p$ for translational motion (and a propulsion torque τ_p for deliberate rotations of the torso) which result from the pedestrian's decision-making process. The LEMONS platform is agnostic to the decision-making model: it leaves free reins to the user to define $\mathbf{F}_p$ and τ_p for each agent as they see fit (see Eq. 4 for a crude proposal). The equation of motion of the centre of mass of agent i (mass m_i, translational velocity $\mathbf{v}_i$) is then expressed as:

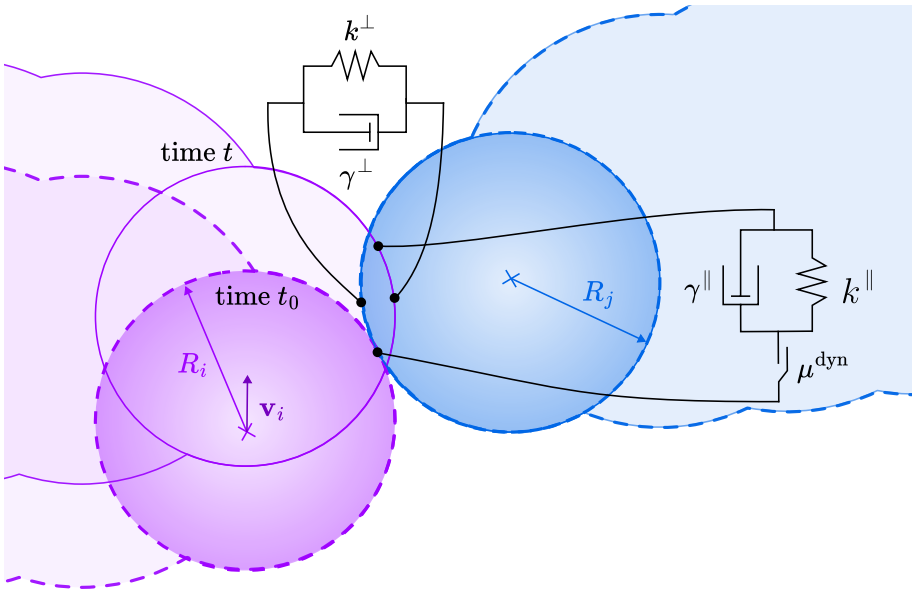

Figure 5: Interactions between composite disks of pedestrians i (radius R_i, velocity $\mathbf{v}_i$) and j (radius R_j, stationary with $\mathbf{v}_j = \mathbf{0}$) are modeled using mechanical elements. In the normal direction (orthogonal to the contact surface), the interaction is described by a spring in parallel with a dashpot (Kelvin–Voigt model), which captures both elastic effects and energy dissipation. In the tangential direction (parallel to the contact surface), the interaction is modelled by a parallel spring-dashpot system in series with a slider, reflecting Coulomb's law. This slider represents a threshold-based element that resists tangential motion until a critical force threshold, proportional to the normal force, is exceeded; after this threshold is reached, it slips at a constant force. The dashed line indicates the pedestrian shape at the onset of contact.

$$m_i \frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} = \mathbf{F}_p - m_i \frac{\mathbf{v}_i}{\mathrm{t}^{(\mathrm{transl})}} + \sum_{(s^{(i)},s^{(j)})\in\mathcal{C}_i^{(\mathrm{ped})}} \left(\mathbf{F}^{\parallel\mathrm{contact}}_{s^{(j)}\to s^{(i)}} + \mathbf{F}^{\perp\mathrm{contact}}_{s^{(j)}\to s^{(i)}}\right) + \sum_{(s^{(i)},w)\in\mathcal{C}_i^{(\mathrm{wall})}} \left(\mathbf{F}^{\parallel\mathrm{contact}}_{w\to s^{(i)}} + \mathbf{F}^{\perp\mathrm{contact}}_{w\to s^{(i)}}\right) \tag{1}$$

where

$$\begin{aligned} \mathcal{C}_i^{(\mathrm{ped})} &= \left\{(s^{(i)},s^{(j)}) \,|\, s^{(j)} \text{ in contact with } s^{(i)}\right\}, \\ \mathcal{C}_i^{(\mathrm{wall})} &= \left\{(s^{(i)},w) \,|\, w \text{ in contact with } s^{(i)}\right\}. \end{aligned} \tag{2}$$

Here, $\mathrm{t}^{(\mathrm{transl})}$ is a characteristic timescale for the effective backwards friction and the $s^{(i)}$ represents the five disks that form agent i. The symbol $\parallel$ indicates a force tangential to the contact surface, while $\perp$ signifies a force orthogonal to the contact surface. These forces are applied to the contact centres, so that they can induce torques on the torso. The rotational dynamics of agent i's torso (moment of inertia I_i, angular velocity ω_i) are governed by:

$$I_i \frac{\mathrm{d}\omega_i}{\mathrm{d}t} = \tau_p - I_i \frac{\omega_i}{\mathrm{t}^{(\mathrm{rot})}} + \sum_{(s^{(j)},s^{(i)})\,\in\,\mathcal{C}_i^{(\mathrm{ped})}} \tau_{G_i,\,s^{(j)}\to s^{(i)}} + \sum_{(s^{(j)},s^{(i)})\,\in\,\mathcal{C}_i^{(\mathrm{wall})}} \tau_{G_i,\,w\to s^{(i)}} \tag{3}$$

where $t^{(\mathrm{rot})}$ is a characteristic timescale for rotational damping and the $\tau_{G_i, s^{(j)} \to s^{(i)}}$ refer to the torque at the centre of mass G_i of pedestrian i, resulting from pedestrian-pedestrian interaction forces. A (very) crude choice for the propulsion force and torque is

$$\mathbf{F}_p = m_i \frac{v^{(0)}\, \mathbf{e}^{(\mathrm{target})}}{t^{(\mathrm{transl})}} \quad \text{and} \quad \tau_p = I_i \frac{-\delta\theta}{(t^{(\mathrm{rot})})^2}, \tag{4}$$

where $v^{(0)}$, $\mathbf{e}^{(\mathrm{target})}$, and $\delta\theta$ are the preferential speed, the unit vector pointing to the target (or way-point), and the angular mismatch between the target direction $\mathbf{e}^{(\mathrm{target})}$ and the front direction of the body of agent i. To solve this set of coupled differential equations of motion, we employed the standard Velocity-Verlet algorithm mentioned in [38], section 3. For additional information on the determination of neighbours, please refer to App. B.

3 The Codebase

The software release consists of (i) an online platform https://lemons.streamlit.app/to generate and visualise individual pedestrians (whose shapes are compatible with anthropometric data) or crowds, (ii) a C++ library to compute mechanical contact forces in two dimensions and then evolve the crowd according to Newton's equation of motion, and (iii) a Python interface to import anthropometric data, generate and visualise crowds, and simulate their dynamics via simple calls to the C++ library. It introduces a generic format for configuration files (which can be saved as XML files) to store agents' shapes and mechanical properties as well as crowds' configurations.

3.1 XML crowd configuration classes

Several levels of detail must be specified to define the configuration of a (presently 2D) crowd, from the geometric and mechanical properties of each of its agents to their positions. We introduce a generic structure made of nested classes, stored as XML files (processed by the third-party library `TinyXML-2`), included in the codebase, to mimic these levels of information; we hope that this structure will be used broadly for the definition of crowd configurations. To illustrate how generic it is, alongside standard adult pedestrians, we will also instantiate geometric objects corresponding to cyclists on their bikes.

Fig. 6 shows how the XML configuration files are used as input and output of the C++ library (which has a Python interface) to simulate the dynamic evolution of the crowd. The contents of each XML configuration file are detailed below. All units are expressed in the International System (SI). One example of each file (used for the practical case example presented in Sec. 4.3) is provided in App. C. We begin with the `Parameters.xml` file:

- **Parameters**:
  - ⋆ Directory in which STATIC files are stored,
  - ⋆ Directory in which DYNAMIC files are stored,
  - ⋆ Time step `TimeStepMechanical` of the Velocity-Verlet algorithm used to solve the dynamics,
  - ⋆ Duration `TimeStep` of one decisional loop, after which $\mathbf{F}_\mathrm{p}$ and τ_p can change, typically a fraction of a second.

"STATIC" files contain information that does not change throughout a simulation, namely:

- **Geometry**:
  - ⋆ `Dimensions` of the simulation area,
  - ⋆ List of 'obstacles' (notably, `wall`), defined as ordered lists of `corners` (i.e., the vertices that are connected by the zero-width wall faces). Each obstacle is made of a given material, whose ID must be specified.

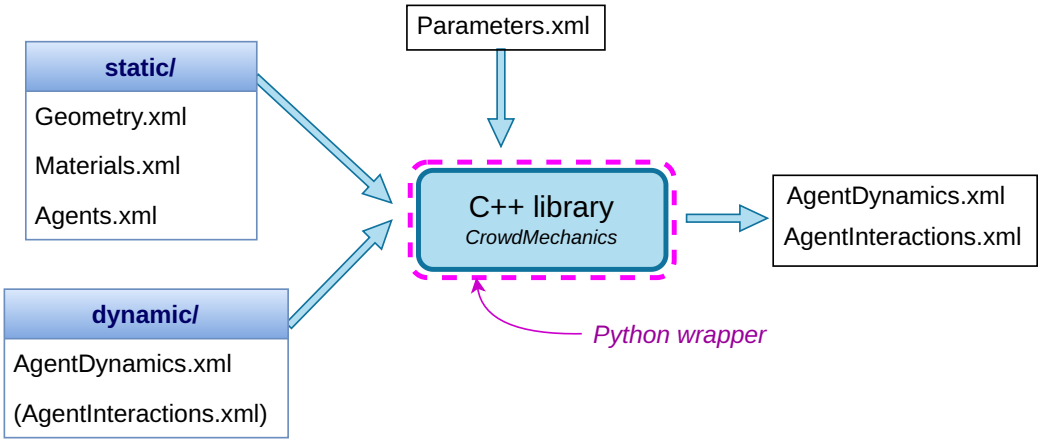

Figure 6: Functional diagram showing the XML configuration files defining the crowd and used as input and output of the mechanical simulation routine, coded in C++ and interfaced with Python.

- **Materials** (for obstacles as well as agents):
  - ⋆ ***Intrinsic* properties**: Young's modulus E, shear modulus G relative to a unique material ID.
    Note: these are used to calculate the spring constants for the contacts between all materials via the following formulas (derived from [39], also see Fig. 5):

$$k^{\perp} = \left(\frac{4G_1 - E_1}{4G_1^2} + \frac{4G_2 - E_2}{4G_2^2} \right)^{-1}, \tag{5}$$

$$k^{\parallel} = \left(\frac{6G_1 - E_1}{8G_1^2} + \frac{6G_2 - E_2}{8G_2^2} \right)^{-1}, \tag{6}$$

  - ⋆ ***Binary* physical properties** that are not reducible to intrinsic ones: damping coefficient $\gamma^{\perp}$ perpendicular to the contact surface, damping coefficient $\gamma^{\parallel}$ tangential to the contact surface, dynamic friction coefficient during slip μ^{dyn}. These need to be defined for all pairs of materials.
- **Agents**:
  - ⋆ ID of the agent,
  - ⋆ `Mass`,
  - ⋆ `Height`,
  - ⋆ `Moment of inertia`,
  - ⋆ Inverse timescale for translational friction $1/\text{t}^{(\text{translational})}$ (`FloorDamping`),
  - ⋆ Inverse timescale for rotational damping $1/\text{t}^{(\text{rotational})}$ (`AngularDamping`),
  - ⋆ Constitutive shapes (5 for a pedestrian):
    - ▷ ID of the constitutive material,
    - ▷ `Radius`,
    - ▷ Initial `position` relative to agent's centre of mass.

    Note: the composite shapes order is important, because the body's orientation will be determined based on the first and last composite shapes. For a pedestrian, the first composite shape should be the left shoulder, and the last one should be the right shoulder.

"DYNAMIC FILES" are used both as input and output of the C++ library, and they contain information that changes during the execution of the code, namely:

- **AgentDynamics** (current state of the agents):
  - ⋆ ***Kinematic*** quantities for each agent:
    - ▷ `Position` **r** of the center of mass,
    - ▷ `Velocity` **v** of the center of mass,
    - ▷ Orientation (`Theta`) of the body concerning the x-axis, that is, the angle θ between the gaze of the agent when looking straight ahead, and the x-axis (see Fig. 11c),
    - ▷ Angular velocity (`Omega`).
  - ⋆ ***Dynamic*** quantities for each agent (not written in the output files):
    - ▷ Propulsion force $\mathbf{F}_\mathrm{p}$ (`Fp`),
    - ▷ Driving torque for the torso τ_p (`Mp`).

  Note: all angular quantities are given relative to the z-axis, with the trigonometric convention.
- **AgentInteractions**:
  - ⋆ Normal force $\mathbf{F}^{\perp\mathrm{contact}}_{s^{(j)}\to s^{(i)}}$ (`Fn`), tangential force $\mathbf{F}^{\parallel\mathrm{contact}}_{s^{(j)}\to s^{(i)}}$ (`Ft`), and tangential spring elongation (`TangentialRelativeDisplacement`) also known as slip (see Sec. A.1.1) between all pairs of composite shapes in contact (that do not belong to the same agent),
  - ⋆ Normal force $\mathbf{F}^{\perp\mathrm{contact}}_{w\to s^{(i)}}$, tangential force $\mathbf{F}^{\parallel\mathrm{contact}}_{w\to s^{(i)}}$, and slip between all pairs (wall – composite shape) in contact.

  Note 1: using the symmetries of forces and spring elongation, we only list the values once for each pair (composite shape – composite shape or wall – composite shape) in contact.

  Note 2: this file is only provided if there are contacts between agents or between agents and walls. No such file is needed in the initial configuration, provided there are no overlaps; for the following runs, the output file can be used *unchanged* for the next run.

3.2 Mechanical layer

In Fig. 6, the mechanical layer *CrowdMechanics* is a C++ shared library that handles the dynamics of the agents described in Sec. 2.2. Calling instructions from C++ and Python are provided in the online tutorials [29].

3.3 Python classes

The Python wrapper mirrors the foregoing structure, insofar as it contains Python classes corresponding to the foregoing XML configuration files. However, since it also allows for generating a synthetic crowd based on anthropometric statistics and visualising it in 2D and in 3D, additional Python classes needed to be defined. The following classes and 'dataclasses' (which contain the statistics and measurements relevant for the generation of the crowd or the agent) are provided:

- ⋆ ***Crowd* class:** group of *Agent* objects.
  The class contains methods to generate a crowd that abides by the measurement constraints of *CrowdMeasures* and to position the agents either on a grid or using the packing algorithm detailed in App. D.
- ⋆ ***CrowdMeasures* dataclass**: Collection of dictionaries representing the characteristics. By default, it contains ANSURII-based anthropometric statistics. But the user can define custom normal distributions for each agent's attributes (e.g., pedestrian bideltoid breadth, bike top tube length).
- ⋆ ***Agent* class**: represents a single pedestrian (or bike rider, etc.)

- ***AgentMeasures* dataclass**: Collection of attribute measurements (e.g., chest depth, mass, height for pedestrians; handlebar length, total bike weight for bikes). The attribute values are taken from *CrowdMeasures* if the agent is instantiated from a *Crowd*; alternatively, they can be specified manually if agents are created one by one.
- ***InitialPedestrian* class**: 2D and 3D contour shapes of a reference pedestrian.
  The 2D shape consists of 5 overlapping discs, whose outer contour matches that of a cryogenic specimen at shoulder's height[2] (see Sec. 2.1 and Fig. 3). There is one 2D template that applies to both men and women, and separate 3D templates: one for men and one for women. To further emphasise the versatility of the file structure, an ***InitialBike* class** was also defined to represent the shape of a rider on a bike, that is to say, a top-down approximate orthogonal projection of the bike and the rider.
  The 3D shape takes the form of a dictionary where each key corresponds to a specific altitude and each value is a `Shapely.MultiPolygon` object. Each `Shapely.MultiPolygon` contains multiple `Shapely.Polygon` objects, each of which is a polygon representing a distinct part of the body—such as a finger, an arm or a leg etc. This is illustrated in Fig. 7.
- ***Shapes2D* class**: 2D shape of a particular agent (pedestrian, bike, ...).
  The 2D pedestrian's shape is obtained by transforming the reference 2D shape (***InitialPedestrian* class**) to match the measurements specified in ***AgentMeasures***. More precisely, the radii of the five disks are uniformly rescaled to match the specified chest depth, defined by the diameter of the middle disk. Additionally, a homothety centred at the agent's centroid is applied to the centres of each of the five composite disks to match the specified bideltoid breadth.
  A similar process is applied for 2D bike shapes; it hinges on the application of homotheties to each composite shape of the reference bike.
- ***Shapes3D* class**: 3D shape of a particular agent (only for pedestrians at present).
  Starting from the reference pedestrian (***InitialPedestrian* class**), the dimensions of each `Shapely.Polygon` at various altitudes are adjusted along with the altitude values themselves. Specifically, a vertical homothety is applied to ensure that the resulting shape matches the desired pedestrian height. Additionally, a homothety is applied to each contour of our reference cryogenic specimen defined in the *InitialPedestrian* class; the centre of the homothety is set at the mean of the x and y coordinates of each polygon's centroid. The scaling factors s_{init} for the homothety are selected to match the chest depth and bideltoid breadth specified in *AgentMeasures*.
  In detail, the chest depth is defined as the maximum distance between two points along the orientation-axis of the `Shapely.MultiPolygon` (i.e., the x-axis if the agent is turned to the right, corresponding to $\theta = 0$) in the slice at torso's height (the altitude used to measure the bideltoid breadth of the cryogenically preserved specimen). The bideltoid breadth is defined as the maximum distance between two points along the axis orthogonal to the orientation-axis of the `Shapely.MultiPolygon` (the y-axis if $\theta = 0$) at the foregoing altitude.
  To avoid inflating the head and feet because of the homothety, the scaling factors $\mathbf{s}_{\text{init}}$ are modulated with the altitude z; the final scaling factor is $\mathbf{s}_{\text{new}}(z) = f(z, \mathbf{s}_{\text{init}})$, where $f(z, \mathbf{s})$ is a smooth, door-shaped function equal to $\mathbf{1}$ for altitudes above the neck and below the knees (meaning no rescaling in those regions) and to $\mathbf{s}$ elsewhere. This approach ensures that, unlike the head, the belly and abdominal regions are duly inflated and reflect morphological differences, particularly for bigger individuals.

[2] More precisely, we consider the horizontal slice at the altitude used to measure bideltoid breadth in our 186.6 cm-tall reference cryogenic male specimen; the same altitude was used to measure the bideltoid breadth of the female specimen (whose feet are extended as if she were on tiptoe, resulting in an elongated posture)

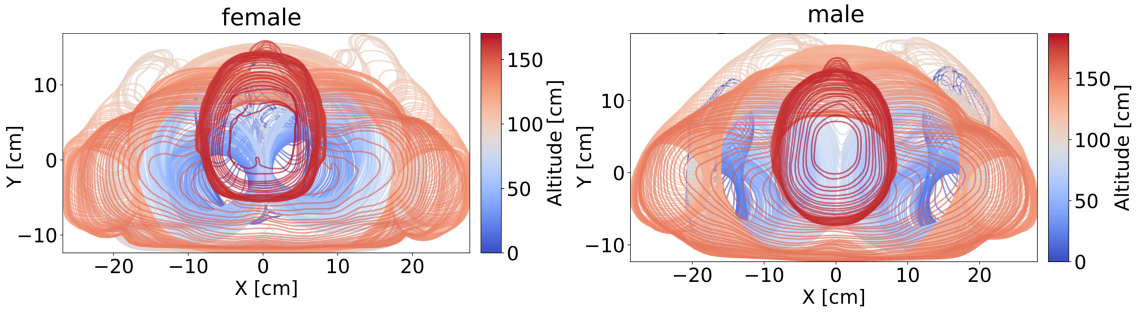

Figure 7: Superimposed cross-sectional contours from two cryogenically preserved bodies, sampled at 0.5 cm intervals. The two bodies belong to a female on the **left** and a male on the **right**. The contours were obtained by processing the images of each section from the dataset [31]. The upper body displays a reddish hue, while the lower part appears bluish.

4 Discussion

4.1 Relevance of the use of 2D projections of standing pedestrians

In line with the dominant approach in pedestrian dynamics, our code primarily operates on 2D shapes, although it provides access to 3D visualisation. This simplification may be questioned not only because pedestrians may raise their arms to protect their chest in very dense settings, but also because people have different heights, so that it may be inadequate to assess their contacts based on 2D projections at torso height. Shorter individuals (e.g., children, women, etc.) may have their heads at the level of the chests or shoulders of taller ones. Consequently, 2D crowd representations can vary significantly depending on the viewpoint of the pedestrian; the practical impact of this question of perspectives remains unclear. Our platform enables us to gauge the extent to which 2D projections reflect the packing conditions in a 3D crowd composed of adults of diverse heights. For this purpose, we generate a static 3D synthetic crowd based on the ANSURII database. In this example, pedestrian heights range from 155 cm to 178 cm for females and from 163 cm to 201 cm for males, with a mean height of 170 cm. As shown in Fig. 8, we present a comparison between the 2D projection of the scene–constructed from our pedestrian shape models–with cross-sections extracted from the corresponding 3D crowd at three distinct altitudes: the torso height of the smallest agent, the torso height of the tallest agent, and the mean torso height across the group. These comparisons reveal that the perceived density can vary considerably depending on the pedestrian's height. Notably, the area covered at the mean torso height is closely matched by our 2D projection approach.

4.2 Mechanical tests

Regarding agent generation, the Pytest files thoroughly test all essential functions, including rotation operations, backup file handling, and file downloading processes. They also ensure that the generated agents' statistical properties—such as mean bideltoid breadth, mean chest depth, and standard deviation—accurately match the intended crowd statistics and anthropometric targets.

Regarding the simulation engine, a series of eight distinct tests (covering distinct scenarios) has been created. You should run these tests each time you modify the C++ code by following the steps below:

1. Navigate to the `tests/mechanical_layer` directory.
2. Run the following command in your terminal:

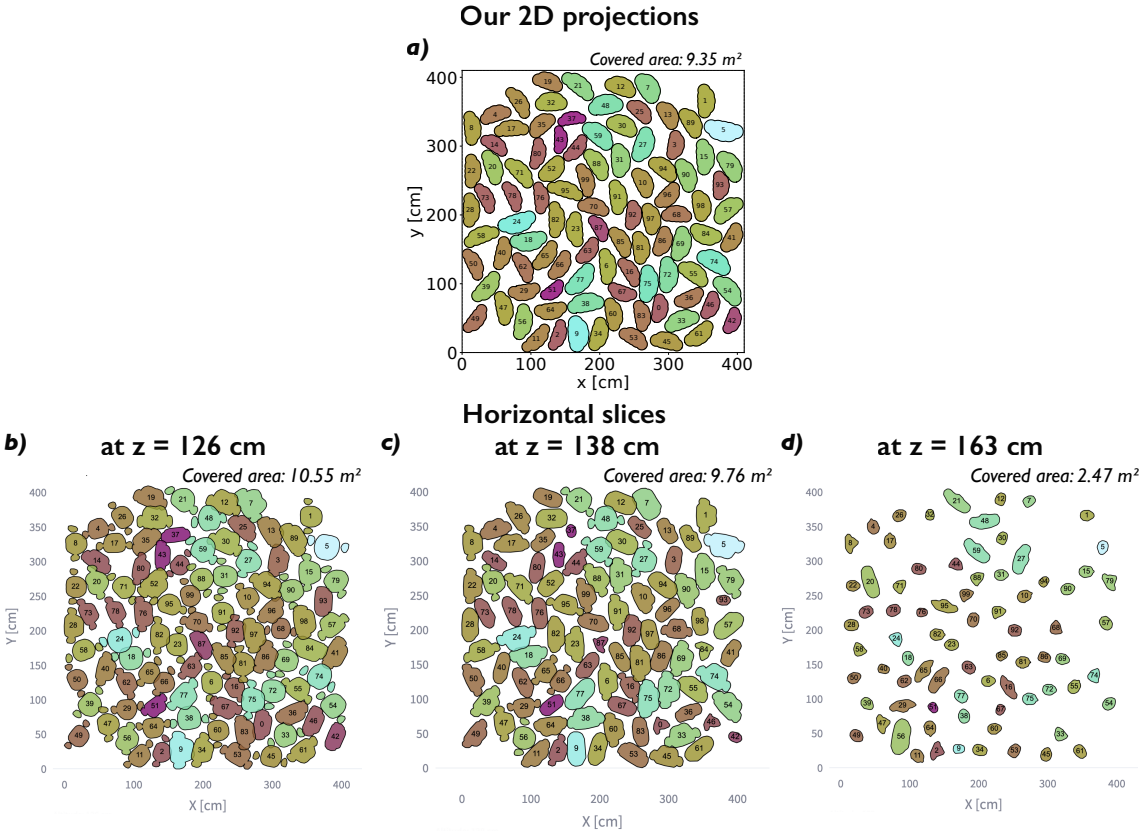

Figure 8: 2D representations of a pedestrian crowd at a density of about 6 ped/m^2. Panel **(a)** shows the 2D projections that we use for the mechanical simulations, while panels **(b)**, **(c)**, and **(d)** display horizontal cross-section of the 3D crowd at altitudes $z = 126$ cm (the torso altitude of the smallest pedestrian in the crowd), $z = 138$ cm (the mean torso height in the crowd), and $z = 163$ cm (the torso height of the tallest pedestrian), respectively. The total areas covered by pedestrian bodies in 2D are indicated.

```
./run_mechanical_tests
```

The script will prompt you to enter the path to your FFmpeg application, which is required to generate movies from the simulation files.

The results of the eight tests will appear in the `tests/mechanical_layer/movies` directory. You will then have the opportunity to review all the videos and determine if they meet your expectations. The eight test scenarios are as follows:

- **Agent pushing another agent** (`test_push_agent_agent` folder)
  Tests the force orthogonal to the contact surface, representing a damped spring interaction between two agents.
- **Agent colliding with a wall** (`test_push_agent_wall` folder)
  Tests the force orthogonal to the contact surface, representing a damped spring interaction between an agent and a wall.
- **Agent sliding over other agents** (`test_slip_agent_agent` folder)
  Tests the Coulomb friction interaction between two agents as one slides over the other.
- **Agent sliding over a wall** (`test_slip_agent_wall` folder)
  Tests the Coulomb friction interaction between an agent and a wall as the agent slides along it.

- **Agent translating and relaxing** (`test_t_translation` folder)
  Tests the behaviour as an agent undergoes a translation and gradually relaxes to a stationary state (no motion), due to the fluid-like force with the damping coefficient of $1/t^{(\text{translation})}$.
- **Agent rotating and relaxing** (`test_t_rotation` folder)
  Tests the behaviour as an agent rotates and gradually relaxes to a stationary state (no motion), due to the fluid-like torque with the damping coefficient of $1/t^{(\text{rotation})}$.
- **Agent rolling over other agents without sliding** (`test_tangential_spring_agent_agent` folder)
  Tests the force tangential to the contact surface, representing a damped spring interaction between two agents.
- **Agent rolling over a wall without sliding** (`test_tangential_spring_agent_wall` folder)
  Tests the force tangential to the contact surface, representing a damped spring interaction between an agent and a wall.

These tests yielded the expected outcomes (the videos are provided in the supplemental material [30]).

4.3 Practical case study

We will detail here how to perform a simulation of a crowd evacuation that achieves *mechanical* realism, albeit (deliberately) naïve as for the decisional component. This foundational example serves as a basis for the future integration of a more advanced decision-making layer (not addressed in this article), which would enable the quantitative replication of empirical observables—such as exit times—recorded in evacuation scenarios. The parameters employed in this simulation are detailed in Tab. 1, and illustrative snapshots can be found in Fig. 9.

Parameter	Description	Value
$t^{(\text{transl})}$	Relaxation time for translational motion	0.5 s
$t^{(\text{rot})}$	Relaxation time for rotational motion	0.5 s
E_{body}	Young modulus for the wall (human naked material)	2.6×10^6 kg/s^2
G_{body}	Shear modulus for the wall (human naked material)	7.5×10^5 kg/s^2
$\gamma^{\perp}_{\text{body}}$	Damping for orthogonal contact in pedestrian-pedestrian contact in the direction orthogonal to the surface contact	1.3×10^4 kg/s
$\gamma^{\parallel}_{\text{body}}$	Damping for tangential contact in pedestrian-pedestrian contact in the direction parallel to the surface contact	1.3×10^4 kg/s
$\mu^{\text{dyn}}_{\text{body}}$	Kinetic friction for pedestrian-pedestrian contact	0.5
E_{wall}	Young modulus for the wall (concrete)	1.7×10^{10} kg/s^2
G_{wall}	Shear modulus for the wall (concrete)	7.1×10^9 kg/s^2
$\gamma^{\perp}_{\text{wall}}$	Damping for orthogonal contact in pedestrian-wall contact in the direction orthogonal to the surface contact	1.3×10^4 kg/s
$\gamma^{\parallel}_{\text{wall}}$	Damping for tangential contact in pedestrian-wall contact in the direction parallel to the surface contact	1.3×10^4 kg/s
$\mu^{\text{dyn}}_{\text{wall}}$	Kinetic friction for pedestrian-wall contact	0.5

Table 1: Parameter values used in the practical case study. The units of the elastic moduli are for 2D systems.

Set the working environment and generate the desired configuration files

Start by creating your desired crowd using the online platform, for example, eight pedestrians with anthropometric characteristics from the ANSURII database, arranged in a tightly packed configuration. Download the resulting configuration files to your local system. For instance,

you can create a new directory called `Trial_1` and navigate into it. Create and configure a `Parameters.xml` file in this directory. Within `Trial_1`, add two subdirectories named `static` and `dynamic`. Place the configuration files obtained from the online platform into their respective folders. For reference, an example of the recommended directory structure is shown below:

```
.
|-- Parameters.xml
|-- static/
|   |-- Agents.xml
|   |-- Geometry.xml
|   |-- Materials.xml
|-- dynamic/
|   |-- AgentDynamics.xml
```

Finally, modify the `Geometry.xml` file to define the desired geometry, and adjust the `AgentDynamics.xml` file to set the appropriate initial propulsion force and torque. Refer to App. C for the configuration files used in this practical case.

Run the simulation

Above all, you need to navigate to the root of the `src/mechanical_layer` directory and build the project:

```
cmake -H. -Bbuild -DBUILD_SHARED_LIBS=ON
cmake --build build
```

Run the Python code provided below, making any necessary modifications to suit your needs. The simulation results will be saved automatically in the `outputXML/` directory. Each output file follows the naming pattern `AgentDynamics output t=TIME_VALUE.xml`, where `TIME_VALUE` indicates the corresponding simulation time or a unique identifier for that run.

```python
import ctypes
from pathlib import Path
import numpy as np
from shutil import copyfile
import xml.etree.ElementTree as ET

# === Simulation Parameters ===
dt = 0.1  # Time step for the decisional layer (matches "TimeStep" in Parameters.xml)
Ndt = 100  # How many dt will be performed in total

# === Paths Setup ===
outputPath = Path("outputXML/")  # Directory to store output XML files
inputPath = Path("inputXML/")  # Directory to store input XML files
outputPath.mkdir(parents=True, exist_ok=True)  # Create directories if they don't exist
inputPath.mkdir(parents=True, exist_ok=True)

# === Loading the External Mechanics Library ===
# Adjust filename for OS (.so for Linux, .dylib for macOS)
Clibrary = ctypes.CDLL("../../src/mechanical_layer/build/libCrowdMechanics.dylib")

agentDynamicsFilename = "AgentDynamics.xml"

# Prepare the list of XML files that will be passed to the DLL/shared library
files = [
    b"Parameters.xml",
    b"Materials.xml",
    b"Geometry.xml",
    b"Agents.xml",
    agentDynamicsFilename.encode("ascii"),  # Convert filename to bytes (required by ctypes)
]
nFiles = len(files)  # Number of configuration files to be passed
filesInput = (ctypes.c_char_p * nFiles)()  # Create a ctypes array of string pointers
filesInput[:] = files  # Populate array with the XML file names

# === Main Simulation Loop ===
for t in range(Ndt):
    print("Looping the Crowd mechanics engine - t=%.1fs..." % (t * dt))

    # 1. Save the current AgentDynamics file as input for this step (can be used for analysis later)
    copyfile("dynamic/" + agentDynamicsFilename, str(inputPath) + rf"/AgentDynamics input t={t * dt:.1f}.xml")

    # 2. Call the external mechanics engine, passing in the list of required XML files
```

```
    Clibrary.CrowdMechanics(filesInput)

    # 3. Save the updated AgentDynamics output to results folder (can be used for analysis later)
    copyfile("dynamic/" + agentDynamicsFilename, str(outputPath) + rf"/AgentDynamics output t={(t + 1) * dt:.1f}.xml")

    # 4. If the simulation produced an AgentInteractions.xml file, save that as well (optional output)
    try:
        copyfile("dynamic/AgentInteractions.xml", str(outputPath) + rf"/AgentInteractions t={(t + 1) * dt:.1f}.xml")
    except FileNotFoundError:
        # If the AgentInteractions file does not exist, skip copying
        pass

    # === Decision/Controller Layer for Next Step ===
    # Read the output AgentDynamics XML as input for the next run.
    # This is where you (or another program) can set new forces/moments for each agent for the next simulation step.
    XMLtree = ET.parse("dynamic/" + agentDynamicsFilename)
    agentsTree = XMLtree.getroot()

    # -- Assign random forces/moments to each agent --
    for agent in agentsTree:
        # Create new <Dynamics> tag for the agent (as the output file doesn't have it)
        dynamicsItem = ET.SubElement(agent, "Dynamics")

        # Assign random force, and random moment
        dynamicsItem.attrib["Fp"] = f"{np.random.normal(loc=200, scale=200):.2f},{np.random.normal(loc=0, scale=50):.2f}
    ↪ "
        dynamicsItem.attrib["Mp"] = f"{np.random.normal(loc=0, scale=5):.2f}"

    # Write the modified XML back, to be used in the next iteration
    XMLtree.write("dynamic/" + agentDynamicsFilename)
    # ================================================

# After all simulation steps are complete, print a final message.
print(f"Loop terminated at t={Ndt * dt:.1f}s!")
```

Generate plots and create a video from output files

A plot of the scene can be generated from each input/output file under PNG format using the Python wrapper. To begin, you need to install the required Python packages, which you can quickly do by setting up a virtual environment using uv as follows (from the root directory of the project):

```
python -m pip install --upgrade pip
pip install uv
uv sync
```

You can then run the following Python script within your working environment:

```
import os
import matplotlib.pyplot as plt
import configuration.backup.dict_to_xml_and_reverse as fun_xml  # For converting XML to dictionary and vice versa
from configuration.models.crowd import create_agents_from_dynamic_static_geometry_parameters  # For creating agents
    ↪ based on XML data
from streamlit_app.plot import plot  # For plotting crowd data

# === Simulation Parameters ===
dt = 0.1  # Time step for the decisional layer (matches "TimeStep" in Parameters.xml)
Ndt = 100  # How many dt will be performed in total

# === Prepare the folders ===
# Define the paths to the folders you'll use
outputPath = Path("outputXML/")
staticPath = Path("./static")
plotsPath = Path("./plots")
plotsPath.mkdir(parents=True, exist_ok=True)  # Create plots directory if it doesn't exist

# Remove any old '.png' files in the plots directory
for file in plotsPath.glob("*.png"):
    os.remove(file)

# === Load static XML files ===
# Read the Agents.xml file as a string and convert it to a dictionary
with open(staticPath / "Agents.xml", encoding="utf-8") as f:
    crowd_xml = f.read()
static_dict = fun_xml.static_xml_to_dict(crowd_xml)

# Read the Geometry.xml file as a string and convert it to a dictionary
with open(staticPath / "Geometry.xml", encoding="utf-8") as f:
    geometry_xml = f.read()
geometry_dict = fun_xml.geometry_xml_to_dict(geometry_xml)

# === Loop over time steps ===
for t in range(Ndt):
    current_time = (t + 1) * dt

    # Check if the dynamics file exists; if not, skip to the next time step
```

```
    dynamics_file = outputPath / f"AgentDynamics output t={current_time:.1f}.xml"
    if not dynamics_file.exists():
        print(f"Warning: {dynamics_file} not found, skipping.")
        continue

    # === Read and process the dynamics XML file ===
    # Read the current dynamics XML file as a string and convert it to a dictionary
    with open(dynamics_file, encoding="utf-8") as f:
        dynamic_xml = f.read()
    dynamic_dict = fun_xml.dynamic_xml_to_dict(dynamic_xml)

    # Create a crowd object using the configuration files data
    crowd = create_agents_from_dynamic_static_geometry_parameters(
        static_dict=static_dict,
        dynamic_dict=dynamic_dict,
        geometry_dict=geometry_dict,
    )

    # Plot and save the crowd as a PNG file
    plot.display_crowd2D(crowd)
    plt.savefig(plotsPath / rf"crowd2D_t={t:d}.png", dpi=300, format="png")
    plt.close()
```

All of the PNG images can then be combined into a video using FFmpeg. Some example snapshots are shown in Fig. 9, and the full video is provided in the supplemental materials [30].

4.4 Extension to arbitrary shapes

This software was designed so that further developments can easily be implemented, particularly to include a wider variety of agents. To prove this point, we implemented bicycles in the 2D agent generation on the online application [40]. To access this feature, navigate to the CROWD tab, then in the sidebar under DATABASE ORIGIN, select the `Custom statistics` option, and set the desired proportion of bicycles within the crowd. The bicycle agent has been simplified to two overlapping rectangular polygons: one representing the front and rear wheels, and the other representing the seated rider and handlebars. The statistics of the dimensions of these shapes are adjustable. Note, however, that the simulation code does not model the mechanical interactions with bicycles, which we consider less relevant and more complex than those between pedestrians. An example of such a heterogeneous crowd is shown in Fig. 10.

The configuration file synthesising the crowd can be downloaded in XML format; it is simpler than the configuration files for a pedestrian-only crowd. The file includes a list of agents, each containing the following information: `type` (either pedestrian or bike), `Id` (an integer), `Moment of inertia` (in kg·m^2), `FloorDamping` ($t^{(\mathrm{transl})}$), `AngularDamping` ($t^{(\mathrm{rot})}$), and `Shapes`. For agents of type `bike`, the `Shapes` tag contains two tags: `bike` (corresponding to the front and rear wheels) and `rider` (corresponding to the human on the bicycle and the handlebars). Within the `bike` tag, several other tags are included: `type` (rectangle), `material` (iron, human clothes, etc.), `min_x`, `min_y`, `max_x`, and `max_y`, which transparently define the rectangle's boundaries in absolute coordinates. The `rider` tag follows a similar structure. For agents of type `pedestrian`, a similar structure is used. However, within the `Shapes` tag, there are sub-tags `disk0`, `disk1`, up to `disk4`, each of which specifies the following attributes: `type` (disk), `radius`, `material`, and `x`, `y` (the position of the disk's centre in absolute coordinates).

5 Conclusion

In summary, we have released an open-source numerical tool to help modellers simulate the dynamics of pedestrians in 2D and visualise the output in 2D and 3D. This tool is *not* a pedestrian simulation software (because the decisional components, notably the desired speeds and directions, should be given as input), but it adds a substantial contribution to the field, especially for the study of dense crowds, in that it promotes realistic 2D projections of pedestrians,

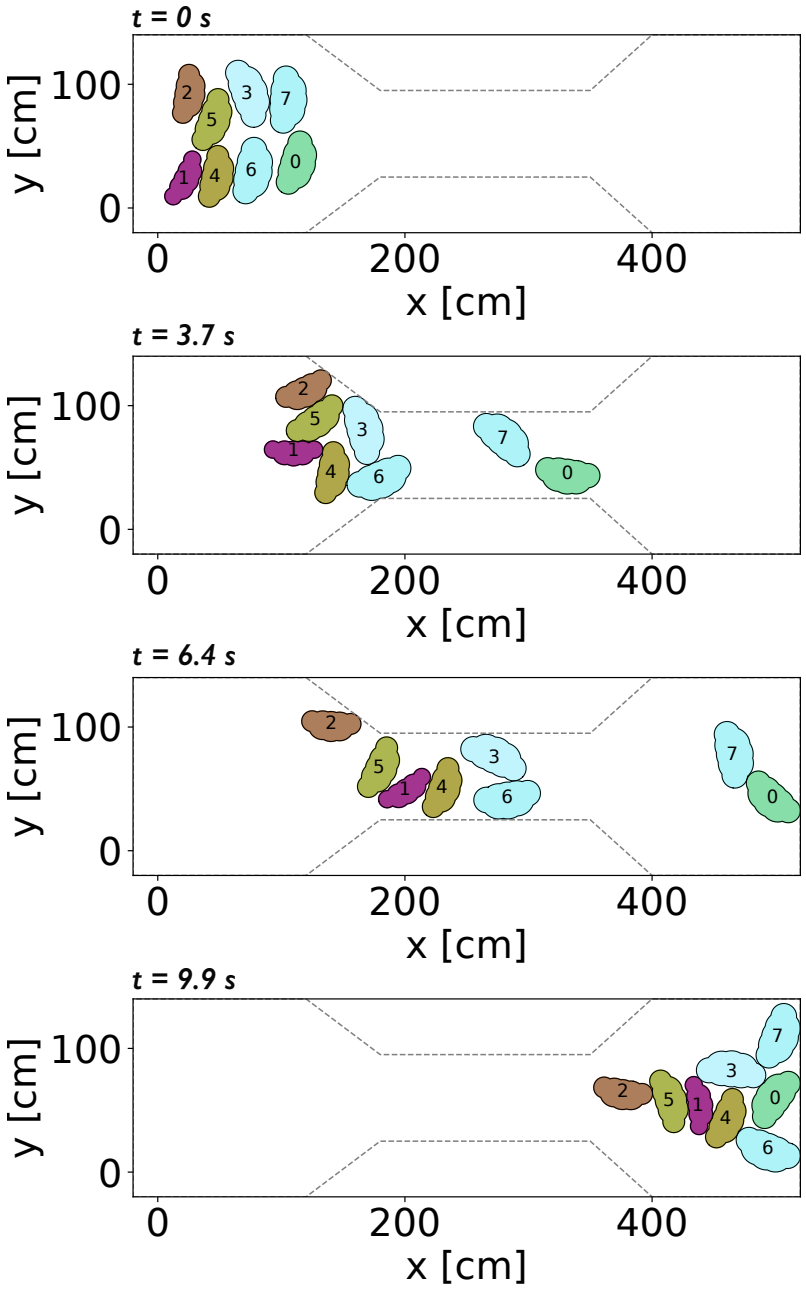

Figure 9: Four snapshots illustrating a crowd simulation in which eight pedestrians move from one room to another through a narrow corridor. The simulation uses a simple decisional model, where each pedestrian's propulsion force $\mathbf{F}_p$ and torque τ_p are randomly sampled from a normal distribution: $\mathbf{F}_{p,x} \sim \mathcal{N}(200, 150)$ | $\mathbf{F}_{p,y} \sim \mathcal{N}(0, 20)$ | $\tau_p \sim \mathcal{N}(0, 10)$. **Top panel:** shows the initial configuration at the start of the simulation. **Second panel (t = 3.7s):** highlights jamming at the corridor entrance; arch-like formations—referred to as 'stress arcs'—have developed, impeding entry into the corridor. **Third panel (t = 6.4s):** the stress arcs have lifted, and pedestrians walk within the corridor almost freely. **Bottom panel:** depicts the final state at the end of the simulation.

grounded in anthropometric data and much more faithful than the typical circular assumption, and it computes contact forces derived from Physics. To make the code as broadly accessible to the public as possible, we have released an online platform for the generation and visual-

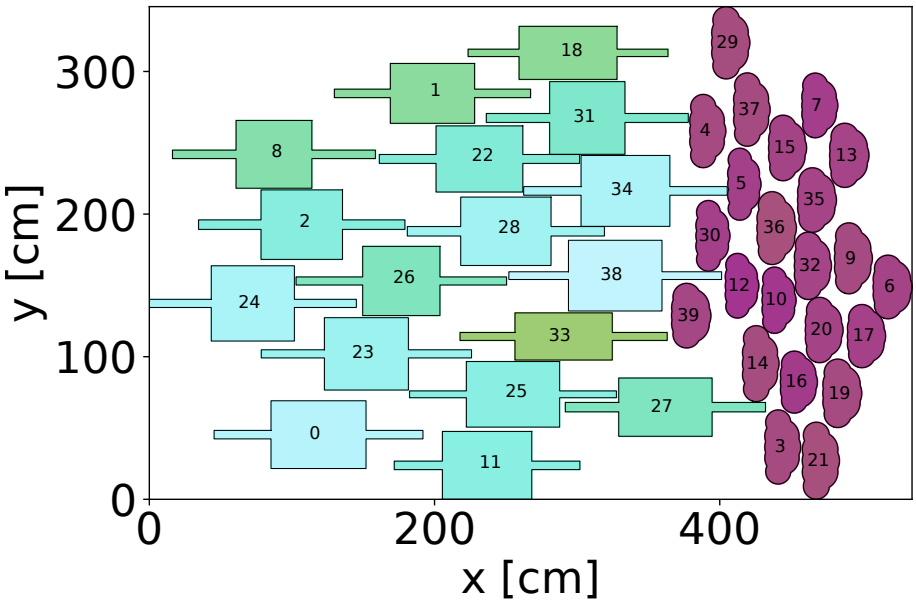

Figure 10: Heterogeneous crowd of 40 agents (17 bicycles + 23 pedestrians) with uniform orientation. The colour encodes agent area using the Hawaii colour map from cmcrameri [41]: transitioning from purple (smallest areas) to blue (largest areas). This example is not intended to be realistic, but rather to showcase that the pedestrian generation code can be easily generalised.

isation of agents, a computationally efficient C++ library for the dynamical simulations, and an easy-to-use Python wrapper to run all scripts.

To let the tool evolve with the field, a generic XML format for configuration files has been proposed. Currently, the tool can only generate bare or clothed adult men and women, as well as cyclists. However, thanks to the generic file format, other shapes may be included in the future, such as children, people carrying a backpack, and people pushing a pushchair. Further in the future, it may also become relevant to extend the mechanical computations of contact forces to 3D.

Acknowledgements

The authors are grateful to David RODNEY for sharing his materials science expertise, to Gaël HUYNH and Mohcine CHRAIBI for their words of advice about code structuring and development.

Author contributions O.D.: Conception, C++ and Python Coding, Testing, Writing –original draft. M.S.: C++ and Python Coding, Testing, Writing –review and editing. A.N.: Conception, Supervision, some Python Coding, Writing –review and editing.

Funding information This work was conducted in the frame of the following projects: French-German research project MADRAS funded in France by the Agence Nationale de la Recherche (grant number ANR-20-CE92-0033), and in Germany by the Deutsche Forschungsgemeinschaft (grant number 446168800), French project MUTATIS funded by Agence Nationale de la Recherche (grant number ANR-24-CE22-0918). This project has also received financial support from the CNRS through the MITI interdisciplinary programs. The authors are not aware

of any competing interests.

A Equation of motion

A.1 Mechanical interactions

Consider two pedestrians, i and j, represented by sets of disks $s^{(i)}$ and $s^{(j)}$ respectively. Each disk center s^i of pedestrian i is positioned relative to pedestrian i's center of mass G_i through the displacement vector $\mathbf{\Delta}_{i\to s^{(i)}}$, which points toward $s^{(i)}$ (see Fig. 11a). The pedestrian's orientation is defined by the normal vector to the line connecting their first and last disks (see Fig. 11c). The CoM of pedestrian i moves with a translational velocity $\mathbf{v}_i$, and the pedestrian rotates with an angular velocity ω_i.

A.1.1 Forces acting on the pedestrian centre of mass

The motion of a pedestrian i can be broken down into two components: the motion of its Center of Mass (CoM) and rotational motion. The motion of the CoM is determined by applying the fundamental principle of dynamics at that point. When the shape $s^{(i)}$ of pedestrian i (with radius $R_{s^{(i)}}$ and position $\mathbf{r}_{s^{(i)}}$) comes into contact with the shape $s^{(j)}$ of pedestrian j (with radius $R_{s^{(j)}}$ and position $\mathbf{r}_{s^{(j)}}$), as illustrated in Fig. 11a, pedestrian i experiences the following forces (analogous forces are applied in the case of contact with a wall, illustrated in Fig. 11b):

- A damped-spring force orthogonal to the surface contact denoted as $\mathbf{F}^{\perp\text{contact}}_{s^{(j)}\to s^{(i)}}$, split into its spring part denoted as $\mathbf{F}^{\perp\text{contact}}_{\text{spring},s^{(j)}\to s^{(i)}}$, linear with the interpenetration depth and a damping part denoted as $\mathbf{F}^{\perp\text{contact}}_{\text{damping},s^{(j)}\to s^{(i)}}$, that can be expressed as:
  - $\mathbf{F}^{\perp\text{contact}}_{\text{spring},s^{(j)}\to s^{(i)}} = \begin{cases} k^{\perp}_{\text{body}}\, h_{s^{(i)}s^{(j)}}\, \mathbf{n}_{s^{(j)}\to s^{(i)}} & \text{if } h_{s^{(i)}s^{(j)}} = R_{s^{(i)}} + R_{s^{(j)}} - |\mathbf{r}_{s^{(j)}\to s^{(i)}}| > 0 \text{ (i.e. an overlap occurs)} \\ \mathbf{0} & \text{otherwise} \end{cases}$
  - $\mathbf{F}^{\perp\text{contact}}_{\text{damping},s^{(j)}\to s^{(i)}} = \begin{cases} -\gamma^{\perp}_{\text{body}}\, \mathbf{v}^{\perp}_{ij} & \text{if } h_{s^{(i)}s^{(j)}} > 0 \text{ (i.e. an overlap occurs)} \\ \mathbf{0} & \text{otherwise} \end{cases}$

  where $\mathbf{n}_{s^{(j)}\to s^{(i)}}$ denotes the unitary vector normal to the surface contact pointing towards $s^{(i)}$, $\mathbf{v}^{\perp}_{ij}$ describes the relative velocity at the contact point C along the direction normal to the surface contact and $\mathbf{r}_{s^{(j)}\to s^{(i)}}$ is the relative position of the two shapes in contact pointing towards shape $s^{(i)}$. $k^{\perp}_{\text{body}}$ represents the spring constant and $\gamma^{\perp}_{\text{body}}$ the damping intensity in the normal direction for body-body contacts.
- A force, tangential to the contact surface that acts in the direction opposite to the slip. A straightforward way to model this force is through Coulomb interaction to describe the stick and slip mechanism, and a damped spring to more precisely describe the stick phase. It can be written as:

$$\mathbf{F}^{\parallel\text{contact}}_{s^{(j)}\to s^{(i)}} = \begin{cases} k^{\parallel}_{\text{body}}\, \delta s \dfrac{-\mathbf{v}^{\parallel}_{ij}}{\left|\mathbf{v}^{\parallel}_{ij}\right|} - \gamma^{\parallel}_{\text{body}} \mathbf{v}^{\parallel}_{ij} & \text{if } k^{\parallel}_{\text{body}}\, \delta s + \gamma^{\parallel}_{\text{body}} \left|\mathbf{v}^{\parallel}_{ij}\right| < \mu^{\text{dyn}}_{\text{body}} \left|\mathbf{F}^{\perp\text{contact}}_{s^{(j)}\to s^{(i)}}\right| \text{ (stick)} \\ \mu^{\text{dyn}}_{\text{body}} \left|\mathbf{F}^{\perp\text{contact}}_{s^{(j)}\to s^{(i)}}\right| \dfrac{-\mathbf{v}^{\parallel}_{ij}}{\left|\mathbf{v}^{\parallel}_{ij}\right|} & \text{otherwise (slip)} \end{cases} \tag{A.1}$$

  where δs represents the spring elongation and can be written as $\delta s = \left| \int_0^{\text{contact duration}} \mathbf{v}^{\parallel}_{ij} \mathrm{d}t \right|$ and $\mu^{\text{dyn}}_{\text{body}}$ denotes the dynamic friction coefficient. The force can be reshaped in a more

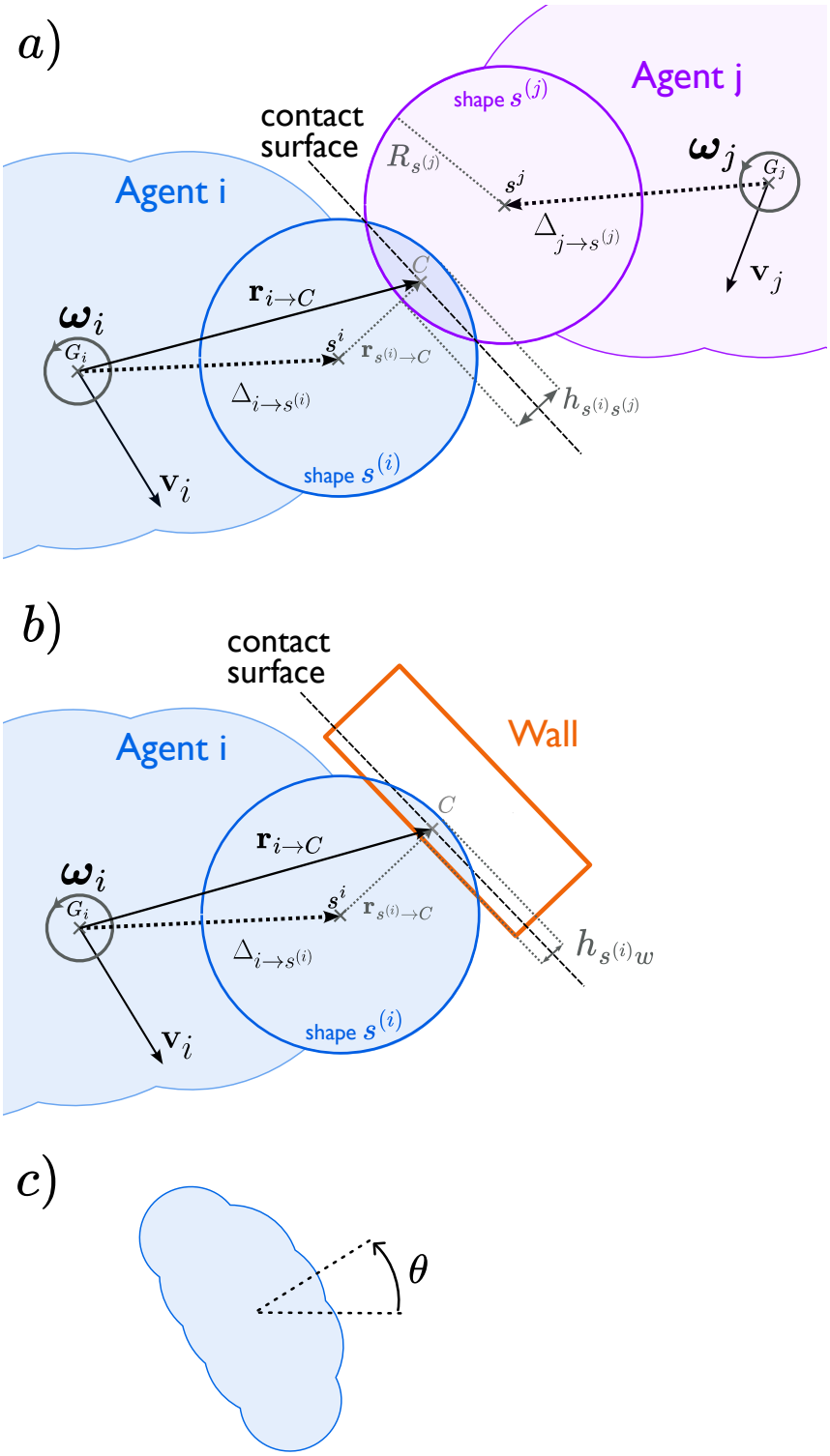

Figure 11: **(a)** Contact between two pedestrian bodies; **(b)** Contact between a pedestrian body and a wall; **(c)** Definition of pedestrian orientation. The contact surface is represented by the bisector of the shortest line segment connecting either the contours of two composite disks or the contour of a composite disk and a wall. The contact point C is located at the midpoint of this segment.

693 condensed way as follows:

$$\mathbf{F}^{\parallel\text{contact}}_{s^{(j)}\to s^{(i)}} = \min\left(k^{\parallel}_{\text{body}}\,\delta s \;+\; \gamma^{\parallel}_{\text{body}}\left|\mathbf{v}^{\parallel}_{ij}\right|\;,\; \mu^{\text{dyn}}_{\text{body}}\left|\mathbf{F}^{\perp\text{contact}}_{s^{(j)}\to s^{(i)}}\right|\right)\frac{-\mathbf{v}^{\parallel}_{ij}}{\left|\mathbf{v}^{\parallel}_{ij}\right|} \tag{A.2}$$

- ⋆ A self-propelling force $\mathbf{F}_p$, that converts decisions into actions;
- ⋆ A fluid friction force, encompassing the effective backward friction with the ground over a simulation step cycle, controlled by the deformation of the body (biomechanical dissipation) expressed as $-m_i\,\mathbf{v}_i/\mathrm{t}^{(\mathrm{transl})}$, where $\mathrm{t}^{(\mathrm{transl})}$ is the characteristic relaxation time to the rest state.

A.1.2 Torque for rotation of a pedestrian

The rotational motion of a pedestrian is obtained by applying the angular momentum theorem to the pedestrian's Center of Mass (CoM). This is done in its principal inertia base, projected along the z-axis (the out-of-plane axis). The pedestrian experiences torque due to the forces that are normal and tangential to the contact surface:

$$\tau_{G_i,s^{(j)}\to s^{(i)}} = \left\{\mathbf{r}_{i\to C}\times\left(\mathbf{F}^{\|\mathrm{contact}}_{s^{(j)}\to s^{(i)}}+\mathbf{F}^{\perp\mathrm{contact}}_{s^{(j)}\to s^{(i)}}\right)\right\}\cdot\mathbf{u_z} \tag{A.3}$$

The self-propelling force and the fluid friction force act directly on the CoM, resulting in zero torque. To account for decision-making, a decisional torque τ_p is applied. Finally, analogous to the CoM equation, a fluid friction force accounting for floor contact and all mechanical dissipation mechanisms (including biomechanical effects) is incorporated as $-I_i\,\omega_i/\mathrm{t}^{(\mathrm{rot})}$. The computation of the moment of inertia I_i is detailed in App. A.2.

A.2 Moment of inertia calculation

Each pedestrian in our synthetic crowd is represented as a combination of five disks. While an analytical formula for the moment of inertia of such a configuration can be derived, it is quite cumbersome to write and implement numerically. Instead, we approximate the pedestrian's boundary using an N-sided polygon, defined by the set of vertices:

$$\{(x_1,y_1),(x_2,y_2),\ldots,(x_{N+1},y_{N+1}) : (x_1,y_1)=(x_{N+1},y_{N+1})\}\,, \tag{A.4}$$

where $(x_1,y_1)=(x_{N+1},y_{N+1})$ ensures the polygon is closed. Assuming pedestrian i's mass m_i is uniformly distributed within the polygon (yielding homogeneous mass density $\rho_i = m_i/\text{Polygon Area}$), the moment of inertia I_i can be calculated via [42]:

$$I_i = \frac{\rho_i}{12}\sum_{j=1}^{N}\left(x_j y_{j+1}-x_{j+1}y_j\right)\left(x_j^2+x_jx_{j+1}+x_{j+1}^2+y_j^2+y_jy_{j+1}+y_{j+1}^2\right). \tag{A.5}$$

A.3 Mechanical equations summary

Pedestrian CoM dynamics

$$\begin{aligned} m_i\frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} = \mathbf{F}_p - m_i\frac{\mathbf{v}_i}{\mathrm{t}^{(\mathrm{transl})}} &+ \sum_{(s^{(j)},s^{(i)})\,\in\,\mathcal{C}_i^{(\mathrm{ped})}}\left(\mathbf{F}^{\|\mathrm{contact}}_{s^{(j)}\to s^{(i)}}+\mathbf{F}^{\perp\mathrm{contact}}_{s^{(j)}\to s^{(i)}}\right)\\ &+ \sum_{(w,s^{(i)})\,\in\,\mathcal{C}_i^{(\mathrm{wall})}}\left(\mathbf{F}^{\|\mathrm{contact}}_{w\to s^{(i)}}+\mathbf{F}^{\perp\mathrm{contact}}_{w\to s^{(i)}}\right) \end{aligned} \tag{A.6}$$

Interaction forces with a pedestrian

$$
\begin{aligned}
\mathbf{F}^{\parallel\text{contact}}_{s^{(j)}\to s^{(i)}} &= \min\left(k^{\parallel}_{\text{body}}\ \delta s\ +\ \gamma^{\parallel}_{\text{body}}\left|\mathbf{v}^{\parallel}_{ij}\right|\ ,\ \mu^{\text{dyn}}_{\text{body}}\left|\mathbf{F}^{\perp\text{contact}}_{s^{(j)}\to s^{(i)}}\right|\right)\frac{-\mathbf{v}^{\parallel}_{ij}}{\left|\mathbf{v}^{\parallel}_{ij}\right|}\\
\mathbf{F}^{\perp\text{contact}}_{s^{(j)}\to s^{(i)}} &= \mathbf{F}^{\perp\text{contact}}_{\text{spring},s^{(j)}\to s^{(i)}} + \mathbf{F}^{\perp\text{contact}}_{\text{damping},s^{(j)}\to s^{(i)}}\\
\mathbf{F}^{\perp\text{contact}}_{\text{spring},s^{(j)}\to s^{(i)}} &= \begin{cases} k^{\perp}_{\text{body}}\ h_{s^{(i)}s^{(j)}}\ \mathbf{n}_{s^{(j)}\to s^{(i)}} & \text{if } h_{s^{(i)}s^{(j)}} > 0 \text{ (i.e. an overlap occurs)}\\ \mathbf{0} & \text{otherwise}\end{cases}\\
\mathbf{F}^{\perp\text{contact}}_{\text{damping},s^{(j)}\to s^{(i)}} &= \begin{cases} -\gamma^{\perp}_{\text{body}}\ \mathbf{v}^{\perp}_{ij} & \text{if } h_{s^{(i)}s^{(j)}} > 0 \text{ (i.e. an overlap occurs)}\\ \mathbf{0} & \text{otherwise}\end{cases}
\end{aligned}
\tag{A.7}
$$

where

$$
\begin{aligned}
h_{s^{(i)}s^{(j)}} &= R_{s^{(i)}} + R_{s^{(j)}} - \left|\mathbf{r}_{s^{(i)}\to s^{(j)}}\right|\\
\delta s &= \left|\int_0^{\text{contact duration}} \mathbf{v}^{\parallel}_{ij}\,\mathrm{d}t\right|\\
\mathbf{v}^{\parallel}_{ij} &= \mathbf{v}_{ij} - \mathbf{v}^{\perp}_{ij}\\
\mathbf{v}^{\perp}_{ij} &= \left(\mathbf{v}_{ij}\cdot\mathbf{n}_{s^{(i)}\to s^{(j)}}\right)\mathbf{n}_{s^{(i)}\to s^{(j)}}\\
\mathbf{v}_{ij} &= \mathbf{v}_{i,C} - \mathbf{v}_{j,C}\\
\mathbf{v}_{i,C} &= \mathbf{v}_i + \omega_i\times\mathbf{r}_{i\to C}\\
\mathbf{r}_{i\to C} &= \boldsymbol{\Delta}_{i\to s^{(i)}} + \mathbf{r}_{s^{(i)}\to C}\\
\mathbf{n}_{s^{(i)}\to s^{(j)}} &= \frac{\mathbf{r}_{s^{(i)}\to s^{(j)}}}{\left|\mathbf{r}_{s^{(i)}\to s^{(j)}}\right|}\\
\mathbf{r}_{s^{(i)}\to s^{(j)}} &= \mathbf{r}_j + \boldsymbol{\Delta}_{j\to s^{(j)}} - \left(\mathbf{r}_i + \boldsymbol{\Delta}_{i\to s^{(i)}}\right)\\
\mathbf{r}_{s^{(i)}\to C} &= \left(R_{s^{(i)}} - \frac{h_{s^{(i)}s^{(j)}}}{2}\right)\mathbf{n}_{s^{(i)}\to s^{(j)}}
\end{aligned}
\tag{A.8}
$$

Interaction forces with wall

$$
\begin{aligned}
\mathbf{F}^{\parallel\text{contact}}_{w\to s^{(i)}} &= \min\left(k^{\parallel}_{\text{wall}}\ \delta s_w\ +\ \gamma^{\parallel}_{\text{wall}}\left|\mathbf{v}^{\parallel}_{s^{(i)}w}\right|\ ,\ \mu^{\text{dyn}}_{\text{wall}}\left|\mathbf{F}^{\perp\text{contact}}_{w\to s^{(i)}}\right|\right)\frac{-\mathbf{v}^{\parallel}_{iw}}{\left|\mathbf{v}^{\parallel}_{iw}\right|}\\
\mathbf{F}^{\perp\text{contact}}_{w\to s^{(i)}} &= \mathbf{F}^{\perp\text{contact}}_{\text{spring},w\to s^{(i)}} + \mathbf{F}^{\perp\text{contact}}_{\text{damping},w\to s^{(i)}}\\
\mathbf{F}^{\perp\text{contact}}_{\text{spring},w\to s^{(i)}} &= \begin{cases} k^{\perp}_{\text{wall}}\ h_{s^{(i)}w}\ \mathbf{n}_{w\to s^{(i)}} & \text{if } h_{s^{(i)}w} > 0 \text{ (i.e. an overlap occurs)}\\ \mathbf{0} & \text{otherwise}\end{cases}\\
\mathbf{F}^{\perp\text{contact}}_{\text{damping},w\to s^{(i)}} &= \begin{cases} -\gamma^{\perp}_{\text{wall}}\ \mathbf{v}^{\perp}_{iw} & \text{if } h_{s^{(i)}w} > 0 \text{ (i.e. an overlap occurs)}\\ \mathbf{0} & \text{otherwise}\end{cases}
\end{aligned}
\tag{A.9}
$$

where

$$
\begin{aligned}
h_{s^{(i)}w} &= R_{s^{(i)}} - |\mathbf{r}_{s^{(i)}\to w}| \\
\delta s_w &= \left| \int_0^{\substack{\text{contact} \\ \text{duration}}} \mathbf{v}_{iw}^{\parallel} \mathrm{d}t \right| \\
\mathbf{v}_{iw}^{\parallel} &= \mathbf{v}_{i,C} - \mathbf{v}_{iw}^{\perp} \\
\mathbf{v}_{iw}^{\perp} &= \left(\mathbf{v}_{i,C} \cdot \mathbf{n}_{s^{(i)}\to w}\right) \mathbf{n}_{s^{(i)}\to w} \\
\mathbf{v}_{i,C} &= \mathbf{v}_i + \omega_i \times \mathbf{r}_{i\to C} \\
\mathbf{n}_{s^{(i)}\to w} &= \frac{\mathbf{r}_{s^{(i)}\to w}}{|\mathbf{r}_{s^{(i)}\to w}|} \\
\mathbf{r}_{i\to C} &= \boldsymbol{\Delta}_{i\to s^{(i)}} + \mathbf{r}_{s^{(i)}\to C} \\
\mathbf{r}_{s^{(i)}\to C} &= \left(R_{s^{(i)}} - \frac{h_{s^{(i)}w}}{2}\right) \mathbf{n}_{s^{(i)}\to w} \\
\mathbf{r}_{s^{(i)}\to w} &= \text{the vector from the center of } s^{(i)} \text{ to its nearest point on the wall } w
\end{aligned}
\tag{A.10}
$$

Rotational dynamics

$$
\begin{aligned}
I_i \frac{\mathrm{d}\omega_i}{\mathrm{d}t} = \tau_p - I_i \frac{\omega_i}{\mathrm{t}^{(\mathrm{rot})}} &+ \sum_{(s^{(j)}, s^{(i)}) \in \mathcal{C}_i^{(\mathrm{ped})}} \tau_{G_i, s^{(j)}\to s^{(i)}} \\
&+ \sum_{(s^{(j)}, s^{(i)}) \in \mathcal{C}_i^{(\mathrm{wall})}} \tau_{G_i, w\to s^{(i)}}
\end{aligned}
\tag{A.11}
$$

Torques

$$
\begin{aligned}
\tau_{G_i, s^{(j)}\to s^{(i)}} &= \left\{ \mathbf{r}_{i\to C} \times \left(\mathbf{F}_{s^{(j)}\to s^{(i)}}^{\parallel\text{contact}} + \mathbf{F}_{s^{(j)}\to s^{(i)}}^{\perp\text{contact}} \right) \right\} \cdot \mathbf{u}_{\mathbf{z}} \\
\tau_{G_i, w\to s^{(i)}} &= \left\{ \mathbf{r}_{i\to C} \times \left(\mathbf{F}_{w\to s^{(i)}}^{\parallel\text{contact}} + \mathbf{F}_{w\to s^{(i)}}^{\perp\text{contact}} \right) \right\} \cdot \mathbf{u}_{\mathbf{z}}
\end{aligned}
\tag{A.12}
$$

B Mechanical layer: agent shortlisting

In order to save computing power, the mechanical layer begins by identifying a subset of the agents, dubbed the "mechanically active agents", for which a collision is likely/possible. The remaining agents are thereby considered as having no chance to collide with anything else during the execution of the code, and will therefore see their position evolve according to the "relaxation" part of equation (1) only. The shortlisting is performed in two steps:

(i) For each agent i, we establish a list of *neighbouring* agents and walls based on

- the radius R_i of the agent – that is, the radius of the circle $\mathcal{C}_i$ centred on the agent's centre of mass, of which the agent's global shape is circumscribed;
- a global constant: the maximum – running – speed $v_{\max} = 7\,\mathrm{m/s}$ of a pedestrian.

Agent neighbours of i will be defined as agents j for which the smallest distance between the borders of the circles $\mathcal{C}_i$ and $\mathcal{C}_j$ is smaller than the distance traveled by both agents at speed $v_{\max}$ in a time `TimeStep` (ie twice the distance traveled at speed $v_{\max}$ in a time `TimeStep`).

Wall neighbours of i will be defined as walls for which the smallest distance between the border of the circle $\mathcal{C}_i$ and the wall is smaller than the distance travelled at speed $v_{\max}$ in a time `TimeStep`).

(ii) We look at new positions of all agents after a uniform motion over time `TimeStep`, with velocity and angular velocity equal to

$$v^{(0)}\,\mathbf{e}^{(\text{target})} = \frac{\mathbf{F}_\text{p}}{m_i}\,t^{(\text{transl})} \quad \text{and} \quad \omega^{(0)} = \frac{\tau_\text{p}}{I_i}\,t^{(\text{rot})},$$

and check for overlaps with neighbours. In case of overlap with a *wall neighbour*, the agent is considered "mechanically active", and in case of an overlap with an *agent neighbour*, both agents are considered "mechanically active". Furthermore, at the end of this process, we also add the *agent neighbours* of "mechanically active" agents.

Finally, agents with a significant difference between the three velocity components above and the ones of their current state – i.e. above 1 cm/s, are added to the list.

C Configuration files example

`Parameters.xml` file

```xml
<?xml version="1.0" encoding="utf-8"?>
<Parameters>
    <Directories Static="./static/" Dynamic="./dynamic/"/>
    <Times TimeStep="0.1" TimeStepMechanical="5e-6"/>
</Parameters>
```

`Geometry.xml` file

```xml
<?xml version="1.0" encoding="utf-8"?>
<Geometry>
    <Dimensions Lx="5.40" Ly="1.60"/>
    <Wall Id="0" MaterialId="concrete">
        <Corner Coordinates="-0.20,-0.20"/>
        <Corner Coordinates="1.20,-0.20"/>
        <Corner Coordinates="1.80,0.25"/>
        <Corner Coordinates="3.50,0.25"/>
        <Corner Coordinates="4.0,-0.2"/>
        <Corner Coordinates="5.20,-0.2"/>
        <Corner Coordinates="5.20,1.4"/>
        <Corner Coordinates="4.0,1.4"/>
        <Corner Coordinates="3.50,0.95"/>
        <Corner Coordinates="1.80,0.95"/>
        <Corner Coordinates="1.20,1.40"/>
        <Corner Coordinates="-0.20,1.40"/>
        <Corner Coordinates="-0.20,-0.20"/>
    </Wall>
</Geometry>
```

`Materials.xml` file

```xml
<?xml version="1.0" encoding="utf-8"?>
<Materials>
    <Intrinsic>
        <Material Id="concrete" YoungModulus="1.70e+10" ShearModulus="7.10e+09"/>
        <Material Id="human_clothes" YoungModulus="3.05e+06" ShearModulus="1.02e+06"/>
        <Material Id="human_naked" YoungModulus="2.60e+06" ShearModulus="7.50e+05"/>
    </Intrinsic>
    <Binary>
        <Contact Id1="concrete" Id2="concrete" GammaNormal="1.30e+04" GammaTangential="1.30e+04" KineticFriction="0.50"
    ↪ />
        <Contact Id1="concrete" Id2="human_clothes" GammaNormal="1.30e+04" GammaTangential="1.30e+04" KineticFriction="
    ↪ 0.50"/>
        <Contact Id1="concrete" Id2="human_naked" GammaNormal="1.30e+04" GammaTangential="1.30e+04" KineticFriction="
    ↪ 0.50"/>
        <Contact Id1="human_clothes" Id2="human_clothes" GammaNormal="1.30e+04" GammaTangential="1.30e+04"
    ↪ KineticFriction="0.50"/>
        <Contact Id1="human_clothes" Id2="human_naked" GammaNormal="1.30e+04" GammaTangential="1.30e+04" KineticFriction
    ↪ ="0.50"/>
        <Contact Id1="human_naked" Id2="human_naked" GammaNormal="1.30e+04" GammaTangential="1.30e+04" KineticFriction="
    ↪ 0.50"/>
    </Binary>
</Materials>
```

`Agents.xml` file

```xml
<?xml version="1.0" encoding="utf-8"?>
<Agents>
    <Agent Type="pedestrian" Id="0" Mass="90.72" Height="1.83" MomentOfInertia="2.05" FloorDamping="2.00" AngularDamping
    ↪ ="2.00">
        <Shape Type="disk" Radius="0.098" MaterialId="human_naked" Position="-0.017,0.164"/>
```

```
        <Shape Type="disk" Radius="0.134" MaterialId="human_naked" Position="0.010,0.072"/>
        <Shape Type="disk" Radius="0.141" MaterialId="human_naked" Position="0.015,0.000"/>
        <Shape Type="disk" Radius="0.134" MaterialId="human_naked" Position="0.010,-0.072"/>
        <Shape Type="disk" Radius="0.098" MaterialId="human_naked" Position="-0.017,-0.164"/>
    </Agent>
    <Agent Type="pedestrian" Id="1" Mass="68.04" Height="1.75" MomentOfInertia="1.18" FloorDamping="2.00" AngularDamping
      ↪ ="2.00">
        <Shape Type="disk" Radius="0.069" MaterialId="human_naked" Position="-0.017,0.167"/>
        <Shape Type="disk" Radius="0.095" MaterialId="human_naked" Position="0.010,0.073"/>
        <Shape Type="disk" Radius="0.100" MaterialId="human_naked" Position="0.015,0.000"/>
        <Shape Type="disk" Radius="0.095" MaterialId="human_naked" Position="0.010,-0.073"/>
        <Shape Type="disk" Radius="0.069" MaterialId="human_naked" Position="-0.017,-0.167"/>
    </Agent>
    <Agent Type="pedestrian" Id="2" Mass="74.39" Height="1.80" MomentOfInertia="1.39" FloorDamping="2.00" AngularDamping
      ↪ ="2.00">
        <Shape Type="disk" Radius="0.084" MaterialId="human_naked" Position="-0.017,0.156"/>
        <Shape Type="disk" Radius="0.116" MaterialId="human_naked" Position="0.010,0.069"/>
        <Shape Type="disk" Radius="0.121" MaterialId="human_naked" Position="0.015,0.000"/>
        <Shape Type="disk" Radius="0.116" MaterialId="human_naked" Position="0.010,-0.069"/>
        <Shape Type="disk" Radius="0.084" MaterialId="human_naked" Position="-0.017,-0.156"/>
    </Agent>
    <Agent Type="pedestrian" Id="3" Mass="115.67" Height="1.83" MomentOfInertia="2.97" FloorDamping="2.00"
      ↪ AngularDamping="2.00">
        <Shape Type="disk" Radius="0.102" MaterialId="human_naked" Position="-0.017,0.180"/>
        <Shape Type="disk" Radius="0.140" MaterialId="human_naked" Position="0.010,0.079"/>
        <Shape Type="disk" Radius="0.146" MaterialId="human_naked" Position="0.015,0.000"/>
        <Shape Type="disk" Radius="0.140" MaterialId="human_naked" Position="0.010,-0.079"/>
        <Shape Type="disk" Radius="0.102" MaterialId="human_naked" Position="-0.017,-0.180"/>
    </Agent>
    <Agent Type="pedestrian" Id="4" Mass="91.17" Height="1.83" MomentOfInertia="1.89" FloorDamping="2.00" AngularDamping
      ↪ ="2.00">
        <Shape Type="disk" Radius="0.088" MaterialId="human_naked" Position="-0.017,0.165"/>
        <Shape Type="disk" Radius="0.122" MaterialId="human_naked" Position="0.010,0.073"/>
        <Shape Type="disk" Radius="0.127" MaterialId="human_naked" Position="0.015,-0.000"/>
        <Shape Type="disk" Radius="0.122" MaterialId="human_naked" Position="0.010,-0.073"/>
        <Shape Type="disk" Radius="0.088" MaterialId="human_naked" Position="-0.017,-0.165"/>
    </Agent>
    <Agent Type="pedestrian" Id="5" Mass="87.54" Height="1.90" MomentOfInertia="1.90" FloorDamping="2.00" AngularDamping
      ↪ ="2.00">
        <Shape Type="disk" Radius="0.088" MaterialId="human_naked" Position="-0.017,0.173"/>
        <Shape Type="disk" Radius="0.122" MaterialId="human_naked" Position="0.010,0.076"/>
        <Shape Type="disk" Radius="0.127" MaterialId="human_naked" Position="0.015,-0.000"/>
        <Shape Type="disk" Radius="0.122" MaterialId="human_naked" Position="0.010,-0.076"/>
        <Shape Type="disk" Radius="0.088" MaterialId="human_naked" Position="-0.017,-0.173"/>
    </Agent>
    <Agent Type="pedestrian" Id="6" Mass="95.25" Height="1.73" MomentOfInertia="2.33" FloorDamping="2.00" AngularDamping
      ↪ ="2.00">
        <Shape Type="disk" Radius="0.104" MaterialId="human_naked" Position="-0.017,0.168"/>
        <Shape Type="disk" Radius="0.143" MaterialId="human_naked" Position="0.010,0.074"/>
        <Shape Type="disk" Radius="0.149" MaterialId="human_naked" Position="0.015,0.000"/>
        <Shape Type="disk" Radius="0.143" MaterialId="human_naked" Position="0.010,-0.074"/>
        <Shape Type="disk" Radius="0.104" MaterialId="human_naked" Position="-0.017,-0.168"/>
    </Agent>
    <Agent Type="pedestrian" Id="7" Mass="98.88" Height="1.68" MomentOfInertia="2.41" FloorDamping="2.00" AngularDamping
      ↪ ="2.00">
        <Shape Type="disk" Radius="0.104" MaterialId="human_naked" Position="-0.017,0.168"/>
        <Shape Type="disk" Radius="0.143" MaterialId="human_naked" Position="0.010,0.074"/>
        <Shape Type="disk" Radius="0.149" MaterialId="human_naked" Position="0.015,0.000"/>
        <Shape Type="disk" Radius="0.143" MaterialId="human_naked" Position="0.010,-0.074"/>
        <Shape Type="disk" Radius="0.104" MaterialId="human_naked" Position="-0.017,-0.168"/>
    </Agent>
</Agents>
```

AgentDynamics.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<Agents>
    <Agent Id="0">
        <Kinematics Position="1.115,0.362" Velocity="0.00,0.00" Theta="-0.29" Omega="0.00"/>
        <Dynamics Fp="100.0,0.0" Mp="0.00"/>
    </Agent>
    <Agent Id="1">
        <Kinematics Position="0.211,0.235" Velocity="0.00,0.00" Theta="-0.48" Omega="0.00"/>
        <Dynamics Fp="100.0,0.00" Mp="0.00"/>
    </Agent>
    <Agent Id="2">
        <Kinematics Position="0.239,0.920" Velocity="0.00,0.00" Theta="-0.16" Omega="0.00"/>
        <Dynamics Fp="100.0,0.00" Mp="0.00"/>
    </Agent>
    <Agent Id="3">
        <Kinematics Position="0.724,0.929" Velocity="0.00,0.00" Theta="0.35" Omega="0.00"/>
        <Dynamics Fp="100.0,0.0" Mp="0.00"/>
    </Agent>
    <Agent Id="4">
        <Kinematics Position="0.463,0.252" Velocity="0.00,0.00" Theta="-0.21" Omega="0.00"/>
        <Dynamics Fp="100.0,0.0" Mp="0.00"/>
    </Agent>
    <Agent Id="5">
        <Kinematics Position="0.437,0.709" Velocity="0.00,0.00" Theta="-0.36" Omega="0.00"/>
        <Dynamics Fp="100.0,0.0" Mp="0.00"/>
    </Agent>
    <Agent Id="6">
        <Kinematics Position="0.757,0.299" Velocity="0.00,0.00" Theta="-0.20" Omega="0.00"/>
        <Dynamics Fp="100.0,0.0" Mp="0.00"/>
    </Agent>
```

```
    <Agent Id="7">
        <Kinematics Position="1.039,0.875" Velocity="0.00,0.00" Theta="-0.07" Omega="0.00"/>
        <Dynamics Fp="100.0,0.0" Mp="0.00"/>
    </Agent>
</Agents>
```

`AgentInteractions.xml` file for $t = 9.9\,\mathrm{s}$

```
<?xml version="1.0" encoding="utf-8"?>
<Interactions>
    <Agent Id="0">
        <Agent Id="3">
            <Interaction ParentShape="2" ChildShape="4" TangentialRelativeDisplacement="-0.000715705,-0.000640808" Fn="
    ↪ 1.21016,-1.3516" Ft="-0.675798,-0.605078" />
        </Agent>
        <Wall ShapeId="4" WallId="0" CornerId="5" TangentialRelativeDisplacement="6.63116e-21,2.03987e-05" Fn="
    ↪ -10.1248,-1.14704e-14" Ft="-7.61778e-14,5.03933" />
    </Agent>
    <Agent Id="1">
        <Agent Id="5">
            <Interaction ParentShape="2" ChildShape="3" TangentialRelativeDisplacement="0.00015316,-0.000992966" Fn="
    ↪ 10.22,1.57639" Ft="0.788194,-5.11002" />
        </Agent>
        <Agent Id="4">
            <Interaction ParentShape="3" ChildShape="3" TangentialRelativeDisplacement="2.1114e-06,3.56781e-06" Fn="
    ↪ -18.5823,10.9969" Ft="-2.56769,-4.33883" />
        </Agent>
    </Agent>
    <Agent Id="2">
        <Agent Id="5">
            <Interaction ParentShape="0" ChildShape="1" TangentialRelativeDisplacement="-0.000122243,0.00122959" Fn="
    ↪ -6.20337,-0.616724" Ft="-0.308362,3.10168" />
        </Agent>
    </Agent>
    <Agent Id="3">
        <Agent Id="7">
            <Interaction ParentShape="4" ChildShape="4" TangentialRelativeDisplacement="-3.22373e-05,2.52169e-05" Fn="
    ↪ -0.137911,-0.176305" Ft="-0.0881525,0.0689554" />
        </Agent>
    </Agent>
    <Agent Id="4">
        <Agent Id="6">
            <Interaction ParentShape="3" ChildShape="4" TangentialRelativeDisplacement="4.14307e-08,8.3854e-08" Fn="
    ↪ -21.8097,10.7757" Ft="0.10954,0.221705" />
        </Agent>
    </Agent>
    <Agent Id="6">
        <Wall ShapeId="0" WallId="0" CornerId="5" TangentialRelativeDisplacement="2.49307e-22,-5.00582e-06" Fn="
    ↪ -31.8342,-8.49639e-15" Ft="-6.12397e-16,11.838" />
    </Agent>
</Interactions>
```

D Packing algorithm within the streamlit app

The `pack_agents_with_forces` method, detailed in Algorithm 1, simulates the arrangement of agents within a bounded environment by iteratively applying physics-inspired, force-based interactions to resolve overlaps and enforce boundary constraints. Additionally, a temperature-based cooling mechanism is used to gradually reduce the magnitude of rotation, helping the system to stabilise. The algorithm relies on the following forces:

Agent-agent repulsive force

For every pair of agents i and j, a repulsive force is computed that decays exponentially with the distance between their centroids:

$$\mathbf{F}_{i,j}^{\mathrm{rep}} = \begin{cases} e^{-|\mathbf{r}_i - \mathbf{r}_j|/\lambda} \, \dfrac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|} & \text{if } |\mathbf{r}_i - \mathbf{r}_j| > 0 \\ \text{random small vector} & \text{otherwise} \end{cases} \tag{D.1}$$

where $\mathbf{r}_i$ is the centroid of agent i and λ is the repulsion length.

Contact force

Algorithm 1: Agent packing with a force-based algorithm

```
Method pack_agents_with_forces(repulsion_length, desired_direction,
  variable_orientation):
    foreach agent do
        RotateTo(agent, desired_direction)
    end
    T ← 1.0                                              ▷ Initial temperature
    for iteration ← 1 to MAX_NB_ITERATIONS do
        foreach agent i do
            forces ← [0, 0, 0]                           ▷ [x, y, rotation]
            foreach agent j ≠ i do
                forces_xy += repulsive_force(i, j, repulsion_length)   ▷ F_{i,j}^rep
                if overlap(i, j) then
                    forces_xy += contact_force(i, j)     ▷ F_{i,j}^contact
                    forces_rot += rotational_force(T)    ▷ f^rot
                end
            end
            if boundary exists and (agent i is in contact with or outside the boundary)
             then
                forces += boundary_forces(i, T)          ▷ F^bound
            end
            if variable_orientation then
                θ_i ← θ_i + forces_rot                   ▷ Update orientation
            end
            r_new = r_current + forces_xy if valid_position(r_new) then
                r_i ← r_new                              ▷ Update position
            end
        end
        T ← max(0, T − 0.1)                              ▷ Cooling
    end
```

If two agents' shapes overlap, a contact force is applied to push them apart:

$$\mathbf{F}_{i,j}^{\text{contact}} = \begin{cases} k \, \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|} & \text{if } \left|\mathbf{r}_i - \mathbf{r}_j\right| > 0 \\ \text{random small vector} & \text{otherwise} \end{cases} \tag{D.2}$$

where k is the contact intensity.

Rotational force

When rotational dynamics are enabled, a random angular adjustment is applied, scaled by the temperature T of the system:

$$f^{\text{rot}} = \text{Uniform}(-\alpha, \alpha) \cdot T \tag{D.3}$$

where α is the maximum rotational intensity.

Boundary forces

If an agent is outside the boundaries or in contact with a force $\mathbf{F}^{\text{bound}}$ is computed to push it back inside, expressed as the sum of:

- a contact force (as above) between the agent's centroid and the closest point on the boundary of the agent's centroid.
- a rotational force (as above), scaled by the current temperature.

E Writing style

To guarantee that our open-source platform, written in Python and C++, is of high quality and easy to maintain, we use automatic checks called *pre-commit hooks*. These checks are triggered each time code is committed or pushed to the Git repository, allowing us to identify and address issues before any changes are integrated into the codebase. For the Python code, the following tools are used:

- **Ruff [43]:** This tool reviews the code to catch common mistakes and ensures we follow established Python programming practices. It performs *linting*, which involves, among other checks, identifying logical errors, overly complex functions, undeclared variables, and deprecated functions. It also handles *formatting*, which means automatically arranging the code—such as fixing indentation, spacing, and comments—so that it remains consistently structured and easy to read.
- **Mypy [44]:** This tool verifies that the types of variables used in the code are consistent—for example, ensuring that a variable defined as an integer remains an integer throughout the program. It also verifies that the types used in the code match the type information described in the function's documentation (type hints), helping to catch mistakes where the wrong variable might be used or returned.
- **NumPydoc validation [45]:** It ensures each function docstring adheres to a clear and standard format.
- **Pytest:** This tool runs a series of small tests on the Python code (using the command `uv run pytest`). If any part of the code produces unexpected results or fails to work as intended, we are immediately alerted.

For both Python and C++ code, we use:

- **CodeSpell [46]:** It scans all the files and helps catch and fix common misspelt words. It does not check for word membership in a complete dictionary, but instead looks for a set of common misspellings.

For the C++ code, we use:

- **clang-format [47]:** This tool arranges the code according to the specific formatting rules recommended by Google. Additionally, it places curly brackets according to the *Allman* style.
- **clang-tidy [48]:** This tool analyses the C++ code to catch common programming mistakes and potential bugs before the program runs. It identifies issues such as violations of coding style, incorrect use of interfaces (for example, calling functions in the wrong way), and typing errors that can be detected by examining the code without executing it (via static analysis).
- **cpplint [49]:** This tool checks that the code follows all the coding style guidelines recommended by Google for C++.

References

[1] D. Helbing, I. Farkas and T. Vicsek, *Simulating dynamical features of escape panic*, Nature **407**(6803), 487 (2000), doi:10.1038/35035023.

[2] C. Wang, L. Shen and W. Weng, *Modelling physical contacts to evaluate the individual risk in a dense crowd*, Scientific Reports **13**(1), 3929 (2023), doi:10.1038/s41598-023-31148-z.

[3] I. Karamouzas, N. Sohre, R. Narain and S. J. Guy, *Implicit crowds: Optimization integrator for robust crowd simulation*, ACM Transactions on Graphics **36**(4), 136:1 (2017), doi:10.1145/3072959.3073705.

[4] M. J. Seitz, N. W. F. Bode and G. Köster, *How cognitive heuristics can explain social interactions in spatial movement*, J. R. Soc. Interface **13**(121) (2016), doi:10.1098/rsif.2016.0439.

[5] I. Echeverría-Huarte and A. Nicolas, *Body and mind: Decoding the dynamics of pedestrians and the effect of smartphone distraction by coupling mechanical and decisional processes*, Transportation research part C: emerging technologies **157**, 104365 (2023), doi:10.1016/j.trc.2023.104365.

[6] C. C. Gordon, C. L. Blackwell, B. Bradtmiller, J. L. Parham, P. Barrientos, S. P. Paquette, B. D. Corner, J. M. Carson, J. C. Venezia, B. M. Rockwell, M. Mucher and S. Kristensen, *2012 anthropometric survey of u.s. army personnel: Methods and summary statistics*, Tech. rep., Defense Technical Information Center, Databse available at https://ph.health.mil/topics/workplacehealth/ergo/Pages/Anthropometric-Database.aspx (2012).

[7] B. Maury and J. Venel, *A discrete contact model for crowd motion*, ESAIM: Mathematical Modelling and Numerical Analysis **45**(1), 145 (2011), doi:10.1051/m2an/2010035.

[8] D. Helbing, A. Johansson and H. Z. Al-Abideen, *Dynamics of crowd disasters: An empirical study*, Physical Review E—Statistical, Nonlinear, and Soft Matter Physics **75**(4), 046109 (2007), doi:10.1103/PhysRevE.75.046109.

[9] J. M. Pastor, A. Garcimartín, P. A. Gago, J. P. Peralta, C. Martín-Gómez, L. M. Ferrer, D. Maza, D. R. Parisi, L. A. Pugnaloni and I. Zuriguel, *Experimental proof of faster-is-slower in systems of frictional particles flowing through constrictions*, Physical Review E **92**(6), 062817 (2015), doi:10.1103/PhysRevE.92.062817.

[10] A. Nicolas, M. Kuperman, S. Ibañez, S. Bouzat and C. Appert-Rolland, *Mechanical response of dense pedestrian crowds to the crossing of intruders*, Scientific reports **9**(1), 105 (2019), doi:10.1038/s41598-018-36711-7.

[11] A. Garcimartín, J. Pastor, C. Martín-Gómez, D. Parisi and I. Zuriguel, *Evacuation narrow door dataset*, doi:10.34735/ped.2013.9, Data collected during evacuation drills at the University of Navarra, Pamplona, Spain on October 26th, 2013 (2013).

[12] S. Feldmann, J. Adrian and M. Boltes, *Propagation of controlled frontward impulses through standing crowds*, Collective Dynamics **9**, 1–16 (2024), doi:10.17815/CD.2024.148.

[13] L. Rothenburg and R. J. Bathurst, *Numerical simulation of idealized granular assemblies with plane elliptical particles*, Computers and geotechnics **11**(4), 315 (1991), doi:10.1016/0266-352X(91)90015-8.

[14] M. A. Hopkins, *Numerical simulation of systems of multitudinous polygonal blocks*, Report, Cold Regions Research and Engineering Laboratory (U.S.), doi:https://apps.dtic.mil/sti/citations/ADA262556 (1992).

[15] C. Hogue and D. Newland, *Efficient computer simulation of moving granular particles*, Powder Technology **78**(1), 51 (1994), doi:10.1016/0032-5910(93)02748-Y.

[16] J. A. Gallas and S. Sokolowski, *Grain non-sphericity effects on the angle of repose of granular material*, International Journal of Modern Physics B **7**(09n10), 2037 (1993), doi:10.1142/S0217979293002754.

[17] A. Dziugys and B. Peters, *Numerical Simulation of the Motion of Granular Material*, Forschungszentrum Karlsruhe, doi:10.1016/S0045-7825(01)00364-4 (1998).

[18] M. Chraibi, A. Seyfried and A. Schadschneider, *Generalized centrifugal force model for pedestrian dynamics*, Physical Review E **82**, 046111 (2010), doi:10.1103/PhysRevE.82.046111.

[19] S. Narang, A. Best and D. Manocha, *Interactive simulation of local interactions in dense crowds using elliptical agents*, Journal of Statistical Mechanics: Theory and Experiment **2017**(3), 033403 (2017), doi:10.1088/1742-5468/aa58ab.

[20] A. Best, S. Narang and D. Manocha, *Real-time reciprocal collision avoidance with elliptical agents*, In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 298–305. IEEE, doi:10.1109/ICRA.2016.7487148 (2016).

[21] Y. Ma, D. Manocha and W. Wang, *Efficient Reciprocal Collision Avoidance between Heterogeneous Agents Using CTMAT*, In *ACM Conferences*, pp. 1044–1052. International Foundation for Autonomous Agents and Multiagent Systems, doi:10.5555/3237383.3237853 (2018).

[22] P. A. Langston, R. Masling and B. N. Asmar, *Crowd dynamics discrete element multi-circle model*, doi:10.1016/j.ssci.2005.11.007, [Online; accessed 17. Jul. 2025] (2006).

[23] F. Alonso-Marroquín, J. Busch, C. Chiew, C. Lozano and Á. Ramírez-Gómez, *Simulation of counterflow pedestrian dynamics using spheropolygons*, Phys. Rev. E **90**(6), 063305 (2014), doi:10.1103/PhysRevE.90.063305.

[24] I. Echeverría-Huarte, I. Zuriguel and R. C. Hidalgo, *Pedestrian evacuation simulation in the presence of an obstacle using self-propelled spherocylinders*, Physical Review E **102**(1), 012907 (2020), doi:10.1103/PhysRevE.102.012907.

[25] R. C. Hidalgo, D. R. Parisi and I. Zuriguel, *Simulating competitive egress of noncircular pedestrians*, Physical Review E **95**(4), 042319 (2017), doi:10.1103/PhysRevE.95.042319.

[26] B. Talukdar and T. Weiss, *Generalized, Dynamic Multi-agent Torso Crowds*, Proc. ACM Comput. Graph. Interact. Tech. **8**(1), 1 (2025), doi:10.1145/3728303.

[27] J. Cusdin, *Iventis an event mapping software for collaborative geospatial planning*, https://www.iventis.com (2015).

[28] B. Kleinmeier, B. Zönnchen, M. Gödel and G. Köster, *Vadere: An open-source simulation framework to promote interdisciplinary understanding*, Collective Dynamics **4** (2019), doi:10.48550/arXiv.1907.09520.

[29] O. Dufour, M. Stapelle and A. Nicolas, *Lemons documentation*, https://lemons.readthedocs.io/en/latest/source.html.

[30] O. DUFOUR, M. STAPELLE and A. NICOLAS, *Lemons : An open-source platform to generate non-circular, anthropometry-based pedestrian shapes and simulate their mechanical interactions in two dimensions*, doi:10.5281/zenodo.16371833 (2025).

[31] . U.S. National Library of Medicine, *Visible human project dataset*, Database available at https://datadiscovery.nlm.nih.gov/Images/Visible-Human-Project/ux2j-9i9a/about_data (1994, 1995).

[32] C. Fryar, Q. Gu and C. Ogden, *Anthropometric reference data for children and adults: United States, 2007-2010*, Vital and health statistics. Series 11, Data from the National Health Survey (2012), https://www.cdc.gov/nchs/data/series/sr_03/sr03_039.pdf.

[33] S. Feldmann and J. Adrian, *Forward propagation of a push through a row of people*, Saf. Sci. **164**, 106173 (2023), doi:10.1016/j.ssci.2023.106173.

[34] C. K. Kroell, D. C. Schneider and A. M. Nahum, *Impact tolerance and response of the human thorax ii*, SAE Transactions pp. 3724–3762 (1974), https://www.jstor.org/stable/44723986.

[35] B. K. Shurtz, A. M. Agnew, Y.-S. Kang and J. H. Bolte, *Effect of Chestbands on the Global and Local Response of the Human Thorax to Frontal Impact*, Ann. Biomed. Eng. **45**(11), 2663 (2017), doi:10.1007/s10439-017-1895-4.

[36] D. C. Viano, I. V. Lau, C. Asbury, A. I. King and P. Begeman, *Biomechanics of the human chest, abdomen, and pelvis in lateral impact*, Accid. Anal. Prev. **21**(6), 553 (1989), doi:10.1016/0001-4575(89)90070-5, 2629763.

[37] X. Li, W. Song, X. Xu, J. Zhang, L. Xia and C. Shi, *Experimental study on pedestrian contact force under different degrees of crowding*, Saf. Sci. **127**, 104713 (2020), doi:10.1016/j.ssci.2020.104713.

[38] D. R. Vyas, J. M. Ottino, R. M. Lueptow and P. B. Umbanhowar, *Improved velocity-verlet algorithm for the discrete element method*, Computer Physics Communications p. 109524 (2025), doi:10.1016/j.cpc.2025.109524.

[39] W. C. Young, R. G. Budynas and A. M. Sadegh, *Roark's Formulas for Stress and Strain, Eighth Edition*, McGraw-Hill Education, New York, NY, USA, ISBN 978-0-07174247-4, https://jackson.engr.tamu.edu/wp-content/uploads/sites/229/2023/03/Roarks-formulas-for-stress-and-strain.pdf (2012).

[40] O. Dufour, M. Stapelle and A. Nicolas, *Lemons streamlit app*, https://lemons.streamlit.app/ (2025).

[41] F. Crameri, *Scientific colour maps*, doi:10.5281/zenodo.1243862 (2023).

[42] D. Hally, *Analysis of polygonal shapes*, Tech. Rep. ADA183444, Defense Technical Information Center, https://apps.dtic.mil/sti/citations/ADA183444 (1987).

[43] Astral, *Ruff: An extremely fast Python linter and code formatter*, https://docs.astral.sh/ruff/.

[44] J. Lehtosalo and contributors, *Mypy: Optional static typing for Python*, https://mypy.readthedocs.io/en/stable/.

[45] Numpydoc Developers, *Numpydoc Validation: Docstring style and completeness checker*, https://numpydoc.readthedocs.io/en/latest/validation.html.

[46] Codespell Project, *Codespell: Fix common misspellings in text files*, https://github.com/codespell-project/codespell.

[47] LLVM Project, *Clang-Format: Formatting C/C++/Obj-C code*, https://clang.llvm.org/docs/ClangFormat.html.

[48] LLVM Project, *Clang-Tidy: C++ "linter" and static analysis tool*, https://rocm.docs.amd.com/projects/llvm-project/en/latest/LLVM/clang-tools/html/clang-tidy/index.html.

[49] cpplint contributors, *cpplint: Static code checker for C++*, https://github.com/cpplint/cpplint.