Computers & Graphics 87 (2020) 103–110

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

An output-driven approach to design a swarming model for architectural indoor environments ☆

C. D. Tharindu Mathew*, Bedrich Benes, Daniel G. Aliaga

Purdue University, Computer Science 305 N University St West Lafayette, IN 47907, United States

ARTICLE INFO

Article history:
Received 17 October 2019
Revised 14 January 2020
Accepted 9 February 2020
Available online 14 February 2020

Keywords:
Swarms
Architectural indoor environments
Output-driven

ABSTRACT

We introduce a novel tool for designing a swarming behavior model for a set of virtual agents to automatically capture an initially unknown indoor architectural environment. Our key idea is to use an output-driven optimization to create targeted swarming behavior. The input to our model is a simple rectangular proxy of the target area and desired acquisition indicator values. The final outputs are the parameters for a swarming behavior model that is autonomous and decentralized, uses only local exploration, and is robust to agent failure. We show and compare the swarming performance in several simulated environments of up to several hundred square meters, 100 agents, and under various conditions.

© 2020 Elsevier Ltd. All rights reserved.

1. Introduction

Capturing and reconstructing interior or exterior spaces is a challenging task that has been addressed by a variety of passive and active methodologies in computer graphics and in computer vision. One option that has received significant attention is to swarm and to capture the environment with a set of agents (*e.g.,* drones or other mobile agents). While the swarms provide parallelism, they are quite challenging to control automatically and robustly so as to produce a high quality result of complex interior and exterior floor-plans.

We focus on an inverse modeling approach to design an autonomous swarming model specialized to capturing architectural indoor environments. The seminal work of [1] introduced swarming model in computer graphics and allowed to reproduce the basic geometric behavior of flocking birds and schools of fish. Significant progress has been made in imitating through a simple set of swarming rules the behaviors of other insects and animals (*e.g.,* [2]), adding noise, velocity models, and control (*e.g.,* [3,4]), and stochastic models (*e.g.,* [5]). Efficient exploration of unknown environments has been also studied by using mobile agents, and most of such setups assume some form of global communication (*e.g.,* [6] focused on indoor spaces and the flying drone system of [7]). Computational swarming approaches are robust, scalable, and inherently parallel. At its core, a swarming approach assumes each agent to individually follow a simple set of rules, with only local interactions, but globally coordinated behaviors emerges. However, these swarm intelligence frameworks do not provide the behavior needed for swarming a set of agents (or mobile capture devices) for acquiring the geometry of a 3D environment. Moreover, we seek a solution where each agent follows simple rules and no map sharing nor centralized control is needed during acquisition. This approach can also be applied to hundreds of intelligent agents in a simulation (*e.g.,* NPCs in video games), as each agent only processes local information, making it computationally efficient and parallelizable.

Our key idea is to build upon output-driven modeling that has been used for images (*e.g.,* [8,9]), 3D models (*e.g.,* [10]), trees (*e.g.,* [11]) and urban models (*e.g.,* [12–14]). In particular, we use an output-driven optimization to design, during a preprocessing phase, a swarming behavior model that yields a desired behavior (i) for an arbitrary number of mobile agents, (ii) that is able to explore an initially unknown architectural indoor environment without needing to share map information nor have centralized control, and (iii) that obtains a desired multi-viewpoint sampling and coverage as shown in Fig. 1.

Altogether, our swarming-behavior design approach consists of two main components (Fig. 2). First, we define a swarming model that assumes individual agents have a fixed sensing radius within which they can sense the environment (*e.g.,* perform laser scans, capture focused pictures at a suitable resolution) and can sense the presence of other agents but do not need to communicate with them. Our swarming model is built on the intuition that a large number of agents having a set of low-cost sensors (such as IR) can use data fusion to obtain 360° field of view input. Further,

☆ This article was recommended for publication by M. Wimmer.

* Corresponding author.
E-mail addresses: mathewc@purdue.edu (C. D.T. Mathew), bbenes@purdue.edu (B. Benes), aliaga@cs.purdue.edu (D.G. Aliaga).

https://doi.org/10.1016/j.cag.2020.02.003
0097-8493/© 2020 Elsevier Ltd. All rights reserved.

Fig. 1. Swarming Architectural Indoor Environments: Our output-driven approach consists of a novel swarming model component and an optimization component which uses a set of indicators to specify desired acquisition behavior. (a) Visualization of the capture process of a swarm of 100 agents through a 750 m^2 environment. Wall coloring indicates level of sampling (red=high) and shading indicates that the location has been visited. This global map is in fact not visible to the individual agents and it is only for visualization purposes. (b) A depiction of the map of a single agent early on in the capture and (c) is the map of the same agent near the end. During the acquisition, the agents do not actually share map information. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

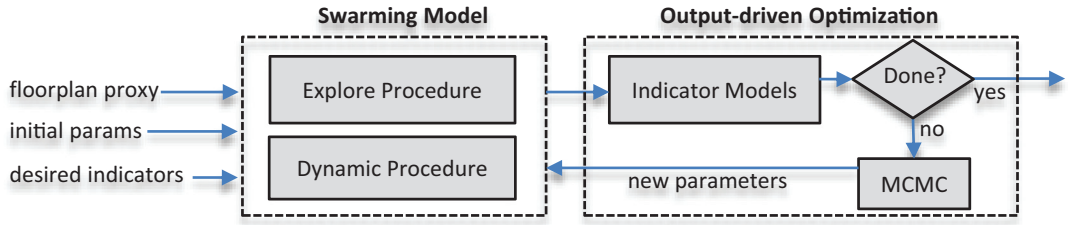

Fig. 2. System Pipeline: The swarming model uses proxies and initial parameters to build the optimization that generates the input parameters of the swarm.

although a large sensor observation radius helps to explore faster, we focus on short-range sensors (*e.g.*, 1 m) to show that it still allows the system to explore the area rapidly when using a swarm. It is common for sophisticated agents (*e.g.*, self-driving cars) to have multiple sensors to cover a 360° field of view. Similarly, many simple ground robots use multiple IR or ultrasound sensors to cover a 360 ° field of view. Hence, we assume the fused sensor data that results in a 360 ° field of view is an input to the swarming algorithm and the sensing distance is not necessarily large. Second, we define an optimization that uses a set of output-behavior indicators (*e.g.*, capture time, coverage, surface sampling, and grouping) to discover the most suitable swarming model parameters. Each agent is only aware of its local surroundings and reacts based only on its neighborhood thus scalable to a large number of agents, though in our current implementation we do address how each agent localizes itself. We do not assume agents need to share exploration maps nor have an initial map of the environment. Only at the end of acquisition would agents coalesce captured data for subsequent processing.

We have analyzed and evaluated our design approach by simulating acquisition in a variety of synthetic 2D floor plans spanning up to several thousand square feet. The swarming behavior models produced by our approach demonstrate to be robust to unexpected and sporadic failure of individual agents throughout the acquisition and are able to dynamically adapt to unexpected geometry and occlusion. Further, we also compare our method to a traditional swarming model [1] and to a multi-agent exploration model with centralized control (*e.g.*, [7]).

We claim the following main contributions: (1) a novel swarming behavior model suitable for robust acquisition of architectural indoor spaces, (2) an output-driven optimization approach for determining swarming behavior model parameters, and (3) a

practical design tool for predicting the suitability of a set of a mobile sensors in a target environment.

2. Related work

Our method builds off of work in multi-agent exploration, computational swarming, and inverse modeling. In terms of 3D reconstruction methods, Seitz et al. [15] compare many multi-view 3D reconstruction methods suitable for a variety of image-based capture mechanisms. Even though, these methods promote the idea of sampling a space with multiple viewpoints, they do not address autonomous swarming-based capture for 3D reconstruction.

Multi-agent exploration and simultaneous localization and mapping (SLAM) focus on constructing a map of an unknown environment while also localizing the agents. Recent work extends solutions to work with multiple agents in a computationally efficient manner (*e.g.,* [16,17]) but the focus is usually on a small number of sensor-heavy devices and often a centralized server. Some methods, such as Fox et al. [18], circumvent a centralized server by sharing map information between nearby agents (and performing a rendezvous operation to mutually register the maps). In contrast, we focus on designing an autonomous decentralized swarming behavior model suitable for many simple agent-like acquisition devices including support for scalability, robustness to agent failure, and simple local control logic. Nonetheless, we anticipate our work can be used in conjunction with other exploration and SLAM methods. Our method allows to explore large maps, where the area is limited only by an agent's memory capacity to hold the map data. Further, simple compression techniques and selective storage (*e.g.,* storing only the points of interest) make use of storage efficient.

Computational swarming is being used in a variety of contexts today. The ability of simple agents to produce complex swarm behavior was demonstrated by Reynolds [1] and Vicsek et al. [3], amongst many others. More recently, Berman et al. [19] and Dorigo et al. [20] study heterogeneous swarms. Kushleyev et al. [21] and Vásárhelyi et al. [22] demonstrated swarming systems of aerial agents carrying out intelligent tasks using a global planner. Rubenstein et al. [23] and Cucu et al. [24] show 1000+ simple agents that operate only on local rules to exhibit global swarm behavior, including making use of the agent itself to form structures. While some swarming models explore unknown environments (*e.g.,* Dirafzoon and Lobaton [25] discover the environment topology), swarming approaches usually do not center on comprehensively mapping an initially unknown environment. A decentralized method are advantageous in hazardous environments, where agents can explore without relying on external server communication and coordination. Our algorithm could steer the agents to complete the task even if some agents were lost.

To date, optimization-based, or output-driven, swarming approaches are less investigated. Some of related prior work in inverse modeling centers on discovering a 3D city configuration that produces a desired set of urban form and function indicator values [12], a 3D parameterized model that corresponds to a simple digital sketch [13], and a 2D walkway layout that yields a desired crowd behavior [14]. Relevant to swarming are data-based approaches (e.g. Wagner and Choset [26]). In particular, the example-based approach of Wang et al. [2] uses the trajectory data of actual swarms to create a swarming model. Also, swarms have been demonstrated by Saska et al. [27] in carrying out a surveillance task for aerial agents. Lee et al. [28] show a swarm robot-based mapping of an environment through a structured triangulation. The algorithm iteratively builds out a triangulation of the environment where each robot acts as a vertex of triangle. Mahadev et al. [29] introduced a particle swarm mapping and coverage algorithm that is controlled by a uniform input to map tissue and vascular systems. In this algorithm each particle experiences the same applied force (such as a magnetic field) as opposed to individual control. Ramachandran et al. [30] introduced a mapping algorithm for a stochastic robot swarm. The dynamics are modeled using a set of advection-diffusion-reaction partial differential equations. The map is incorporated into the model, and the system is solved as a optimization problem offline using a gradient descent algorithm. In contrast to the aforementioned works, our solution seeks to comprehensively explore an unknown environment without a centralized server, to create clusters when desired, to sample surfaces multiple times, and to reduce capture time. At the conclusion of capture, data is coalesced and processed offline. Further note, we are not attempting to imitate swarm behavior in nature but instead acquire environment geometry.

3. Overview

Fig. 2 shows an overview of our system pipeline. The input to our swarming behavior design tool is a simple floor plan proxy (*e.g.,* an approximate 2D floor plan with only major walls and obstacles), the desired acquisition indicator values (*e.g.,* values for simultaneous sampling, coverage and time taken), and an initial set of swarm agent parameters (*e.g.,* number of agents, starting positions). Note that the floor plan proxy is not needed for the swarming model. The output of our design tool are the parameters of our autonomous decentralized swarming behavior model that can be used during an actual acquisition where the agents use our swarming behavior model.

The first block shows the *swarming model* that uses explore and dynamics procedures to move through the input floor plan (Section 4). The *explore procedure* executes steps related to deciding the next location. The dynamics procedure combines five velocity components (i.e., *explore*, *obstacle avoidance*, *alignment*, *cluster* and *separation*) to form the final velocity of motion for each agent a_i, $i = 0, 1, \ldots, |a|$.

The second block describes the *optimization* approach that iteratively modifies the parameters of our swarming model in order to yield agent behaviors that result in the desired indicator values (Section 5). The parameters of the swarming model are the five coefficients of each velocity component which are found via an MCMC-based optimization and numerous acquisition simulations. Each simulation moves the agents throughout a discretized grid, with each grid cell being denoted as g_{xy}.

We assume the agents in our model (and the comparison) have the following characteristics:

1. The motion is restricted to the 2D ground plane
2. The union of sensors of an agent is able to sense objects and agents within a radius of 1m (diameter of 2 m), within a total circular arc, *i.e.,* 360°. The swarming algorithm receives the output after completing sensor fusion (simply, the union of the sensors). This radius of the fused sensor is configurable, but for our experimental purposes it is chosen to have a short-range to emulate a set of low-cost sensors.
3. The average speed of motion of each agent is 1 ms^{-1} (initialized from a normal distribution, $\mathcal{N}(1, 0.1^2)$, with the maximum being 4 ms^{-1}.
4. Velocity updates are bounded and happen at 30 Hz hence restricting the maximum acceleration to beneath a desired value.

4. Swarming model

Our approach designs a swarming behavior model that can efficiently explore and sense an initially unknown architectural indoor environment. We assume each agent will be equipped with

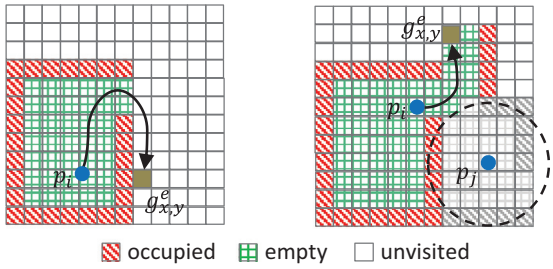

Fig. 3. Explore Procedure: (a) Our procedure uses a combination of breadth-first-search and A* to explore grid cells. (b) If an agent *i* encounters an agent *j*, the other agents sensing area is consider occupied in *is* map and it visits elsewhere. agent *js* map, shown in gray-scale, is not known to agent *i* it simply marks all the gray-scale area as occupied.

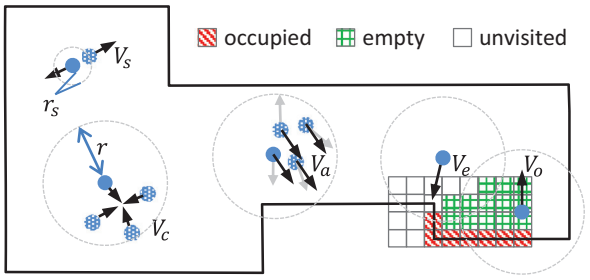

Fig. 4. *Dynamics Procedure.* We depict the five velocity types of our dynamics model and the underlying grid cell and its potential cell flag values.

some form of environment sensing (*e.g.,* a range-detecting system or an image-based system with color and/or depth capture). While the environment does not strictly need to be indoors, we do assume it is bounded and it can be described by an interconnected space of polygon-shaped rooms, corridors, and obstacles that can be modeled with a simple planar interconnection graph. Given a number of agents and a prioritization of the relative importance of minimizing acquisition time, achieving a certain level of surface sampling, a wanted coverage percentage, and a desired amount of grouping, our optimization model computes a suitable set of swarming model parameters. Then, as verification we simulate how the agents explore and navigate through the environment.

4.1. Swarming behavior model

Our approach defines a swarming behavior model that collectively acquires a target environment. In the following, we describe our exploration and dynamics procedures.

Exploration: During our exploration procedure, each agent explores the initially unknown scene with the help of a 2D grid assumed to be big enough to capture the environment. Each grid cell is represented by its centroid; initially all cells are marked as unvisited and later a cell can be marked as empty or occupied. We denote each agent by an index i or j and our per-agent variable notation is as follows: p_i is the position of agent i (in the grid), v_i is the velocity of ith agent, and r_i is the sensing radius of an agent.

All agents start exploration from a tight formation. As each agent i explores the environment, it marks its explored cells as empty (i.e., free space) or occupied (i.e., wall or object as per sensor readings) (Fig. 3). The details of the dynamics procedure are explained below, but essentially either agent i follows agent j to achieve a desired level of simultaneous sampling, or agent i explores on its own to increase sampling coverage. An agent may explore on its own if its explore velocity component overwhelms the others. This is especially true if the agent's map contains unexplored areas that are away from the other agents. If an agent i explores on its own and encounters another agent j, then all grid cells within agent js sensing radius are marked as occupied in agent is grid. If the width of any part of the explorable environment is less than or equal to r and the interconnection graph of the interior spaces is single-connected (i.e., there is no bi-connected subgraph or, intuitively, there is only one pathway to access each room), then the aforementioned logic will prevent agent i from unnecessarily capturing the area already sampled by agent j. In practice, we can relax the single-connected constraint because the presence of other agents $k_1, l_2, \ldots, k_q$ will typically block agent is access to areas sampled by agent j (*e.g.,* it will rarely occur that agent i arrives to the space behind agent j by another route through the environment without encountering other agents. If this were to happen after agent i observes agent j it would explore another location.

The acquisition is complete when the space accessible to each agent is marked as occupied (either through own exploration or occupied by walls/obstacles or occupied by other agents sensing areas).

Dynamics: The dynamics procedure determines a agents motion by making use of a weighted sum of five different velocity types (Fig. 4). During each iteration, all velocities are recomputed for each agent. The iterations terminate when the explore velocity $v_i = 0$ (i.e., there is nothing left to explore). The velocity update for agent i is:

$$\Delta v_i = k_e(f_{V_e}(V_e) - V_i) + k_o f_{V_o}(V_o) + k_a(f_{V_a}(V_a) - V_i) + k_c f_{V_c}(V_c) + k_s f_{V_s}(V_s), \quad (1)$$

where the subscripts e, o, a, c and s correspond to explore, obstacle-avoidance, alignment, cluster, and separation velocities, respectively. The velocity weights k_e, k_o, k_a, k_c and k_s are the primary parameters that determine a agents behavior. $f_{V_e}, f_{V_o}, f_{V_a}, f_{V_c}$, and f_{V_s} are sensitivity functions that scale the effect of the velocity type for the different velocity types.

Inspired by Reynolds [31], each sensitivity function has the form:

$$f(V) = |V|^x \frac{V}{|V|}, \quad (2)$$

but x varies according to the velocity type. For our five types, we found $x_e = 2, x_o = -2, x_a = -2, x_c = -2,$ and $x_s = -2$ to work well. For example, $x_e = 2$ implies that the exploration velocity is more sensitive (i.e., larger) for large distances and $x_s = -2$ implies the separation velocity is larger for small distances. The current velocity is subtracted from the exploration and alignment velocities because those velocities are used in a corrective goal-seeking capacity.

Our procedure uses an explore velocity V_e to wander into new unexplored areas. Selecting goal grid cell g^e_{xy} and computing the corresponding explore velocity vector is performed using a search algorithm. A breadth-first-search (BFS) of the cells in the map of agent i is performed starting at p_i. The closest unvisited cell becomes u_i. The A* search algorithm and the current grid are used to determine a path to u_i. The next grid cell along the A* path to u_i becomes g^e_{xy}. To save on compute time, the A* algorithm, and selection of u_i, is re-computed only when some cell is newly marked as occupied (i.e., the map has changed significantly). Further, for the A* computation, we give a cell adjacent to a wall/obstacle a higher weight than other empty/unknown cells. This encourages the algorithm to compute a path that goes close to the walls/obstacles but not too close (when possible). Then, given g^e_{xy} the explore velocity is:

$$V_e = g^e_{xy} - p_i. \quad (3)$$

In order to avoid walls or objects, our model uses an obstacle avoidance velocity V_o. The computed velocity essentially avoids

hitting (or reaching) the obstacle. One definition of an obstacle avoidance velocity is

$$V_o = \frac{1}{N_i^o} \sum_{\|p_i - g_{xy}^o\| \le r} (p_i - g_{xy}^o), \tag{4}$$

where N_i^o is the number of grid cells marked as occupied within a distance r of p_i and g_{xy}^o represents an occupied grid within a sensor radius. Note that we use ray casting to only include obstacles visible with an unobstructed line of sight. This formula basically computes the averaged vector that moves the agent away from the obstacle. In our implementation, we use a variation of Eq. (4) specific to rectangular grids where given a agent moving nearly parallel to the obstacle, it can continue moving in the same general direction but swerves away from the obstacle. Our method uses an alignment velocity V_a and a cluster velocity V_c to encourage agents to move in unison and to cluster together. It is computed as the average velocity/position (from the previous iteration) of all agents within distance r of p_i. We call this cluster C^i. Thus,

$$V_a = \frac{1}{|C^i|} \sum_{j \in C^i} v_j \tag{5}$$

$$V_c = \frac{1}{|C^i|} \sum_{j \in C^i} (\hat{p}_i - p_j), \tag{6}$$

where $|C^i|$ is the size of cluster C^i and $\hat{p}_i$ is the centroid of cluster C^i.

Finally, to avoid agents colliding with each other, we use a separation velocity V_s. This velocity is inversely proportional to the distance between agent i and other agents within distance $r_n < r$. The separation velocity for agent i is

$$V_s = \sum_{\|p_i - p_j\| \le r_n} (p_i - p_j). \tag{7}$$

This velocity moves agent i away from other agents j. In the rare case agent i is surrounded symmetrically by various agents, agent i will not move - instead the other agents will.

4.2. Discussion

Let us consider the case where a single agent explores an arbitrary floor plan with obstacles. BFS is used to find goal cells, and A* is used to find the path. If we assume the obstacle avoidance value is set to a value that does not repel the agent away before it reaches the cells close to an obstacle, then the explore procedure guarantees that all reachable positions will be explored. If multiple agents are exploring, a pathological case occurs when the cluster velocity (which attracts agents to adjacent agents) overwhelms all other velocities and it results in all agents staying in one place without any exploration. Though theoretically possible, in practice we have not encountered this scenario and instead show (see Section 5) multiple sets of parameters that achieve varying desired outcomes.

5. Optimization model

We use an optimization approach to determine the swarming model parameters that will best yield desired target values for an intuitive set of indicators tailored to 3D acquisition. In the following, we describe the indicators and the optimization method used.

5.1. Simultaneous sampling

We seek to control multiple agents so as to achieve a desired level of simultaneous surface sampling of the scene, denoted as S. Prior work has shown that the quality and robustness of a 3D reconstruction increases with multiple viewpoints. Given a single (mobile) sensor, we could collect sensor readings from multiple viewpoints of each scene surface. In preliminary work we found that if we considered multiple samples over time, agents tended to stay near a grid cell for some time in order to satisfy the multiplicity. Instead, for multiple (mobile) sensors we define simultaneous sampling to be the desired property of acquiring more than one sensor reading of a scene surface at any given time.

Simultaneous samples imply the number of agents for which the cell was within the sensor distance (and in line-of-sight) of a agent during the same iteration of the update loop. We label the maximum number of simultaneous samples of an occupied grid cell g_k to be s_k and it is computed as:

$$s_k = \max_t[\text{count}(|g_{xy} - p_i^t| \le r)], \tag{8}$$

where count() provides a tally of the number of expressions satisfying the conditional, g_{xy} is the grid cell marked as occupied and p_i^t is the position of a agent at time (or iteration) $t \in [0, T]$ where T is the simulation time so far. The total simultaneous sampling for all grid cells is

$$S = \sum_k s_k, \tag{9}$$

Coverage: The user can also control the coverage C of the target environment. This enables performing a tradeoff between coverage and other indicators such as time taken and simultaneous sampling (*e.g.,* agents can be directed to avoid sampling small time-consuming spaces).

The coverage is measured as a ratio between the sampled grid cells marked as empty/occupied (i.e., different than unvisited) and the total number of grid cells:

$$C = \frac{count(g_k \neq unvisited)}{G}, \tag{10}$$

where G is the actual (or estimated) total number of grid cells that should be marked as empty/occupied. The correctness of this coverage quantity depends on the accuracy of the grid cell array. If the grid cell array nearly matches the shape and size of the actual floorplan, then a true coverage indicator is computed. However, in our simulations the grid cell array is a simple proxy (*e.g.,* a rectangle) thus the coverage quantity is only a coarse estimation. Nonetheless, in both cases selecting the coverage level is a beneficial indicator.

Time taken: Time taken T is a measure of how fast the agents achieve a desired scene coverage. The number of time steps keeps increasing until the explore velocities of all agents are zero.

Grouping: We define a grouping indicator G which measures how well defined are groups of agent clusters. This behavior may be beneficial to simultaneous sampling, to agents occupying a more compact footprint, and, in future work, to assist with localized pose and orientation estimation. The grouping indicator at time t is

$$G_t = \frac{\sum_i count(|p_i^t - p_j^t| \le r)}{\text{NN}_i^t} \tag{11}$$

and the overall average grouping indicator is

$$G = \frac{1}{T} \sum G_t, \tag{12}$$

where G_t is the average number of agents within r of each agent i at time step $t \in [1, T]$, N is the total number of agents, and N_i^t is the number of agents within r of agent i at time t. p_i^t and p_j^t are the positions of agents i and j at time t.

5.2. Optimization

The objective function for use during our optimization of the swarming behavior model parameters is formed by a weighted

combination of the aforementioned four indicators:

$$\Phi(\mathrm{S,C,K,T}) = \gamma_s(S - S_d)^2 + \gamma_c(C - C_d)^2 + \gamma_k(G - G_d)^2 + \gamma_t T^2, \tag{13}$$

where S_d, C_d, and G_d are desired indicator values (while usually for time taken the expected value is zero). Combining these four indicators and coefficients, we form the optimization function to minimize as:

$$\mathrm{argmin}_{k_e,k_o,k_a,k_c,k_s}\,\Phi(\mathrm{S,C,K,T}). \tag{14}$$

Since the solution space is nonlinear, a straightforward gradient descent type method will probably fall into a local minimum. Instead, we use a controlled randomized walk methodology, such as a Markov Chain Monte Carlo (MCMC) [32]. In particular, we use the Metropolis Hastings algorithm [33,34] to generate potential state changes using a per-parameter probability density function. This methodology essentially enables us to explore the solution space and, under the right conditions, has in fact been proven to find the optimal solution. Based on experimentation, we found a reasonable MCMC setup using 100 simultaneous chains running at four temperatures and typically for 1000 steps. The normalized objective value reaches a value of 10^{-3} upon convergence. An experimental attempt to reuse a set of fixed values for each map yields sub-par results. These values could always be improved through our optimization approach, due to the fact that obstacle avoidance velocity and exploration velocity can have differing values that respond to differing map characteristics (*e.g.,* open spaces vs thin corridors). Furthermore, the different requirements within the same map such as a combined simultaneous sampling and time taken requirement would need a tedious and time-consuming manual tuning of the parameters.

6. Results and discussion

We have applied our approach to a variety of floorplans in simulation. Our prototype implementation was developed in C/C++ and runs on a PC with 3 GHz Intel i7 Processor, 16 GB RAM, and a NVidia GTX 970 GPU. All visualization is done using custom OpenGL code and optimization is performed using an in-house MCMC engine.

Our swarming behavior model is accelerated by the use of spatial-indexing to quickly determines cells adjacent to a agent. Further, we make use of jump point search [35] to significantly accelerate our A* computations. In practice, our swarming simulator runs a single agent at realistic velocities through a several thousand square foot environment in under one second of computation time and a 50 agent swarm in ten seconds.

Our optimization performs a MCMC-based optimization to find the swarming behavior model parameter values. Currently, to define a fixed relative scale of the solution vector we fix k_o to a constant value and optimize only the other four parameters. On our computer, it takes an average of 5–15 min to compute a solution for a typical floor plan.

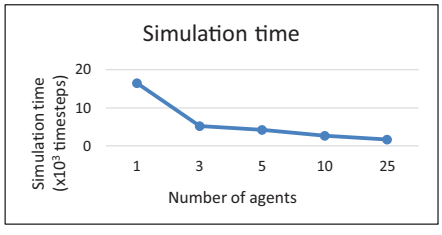

Fig. 5. Simulation time: Solution computed to minimize time taken to sample entire environment.

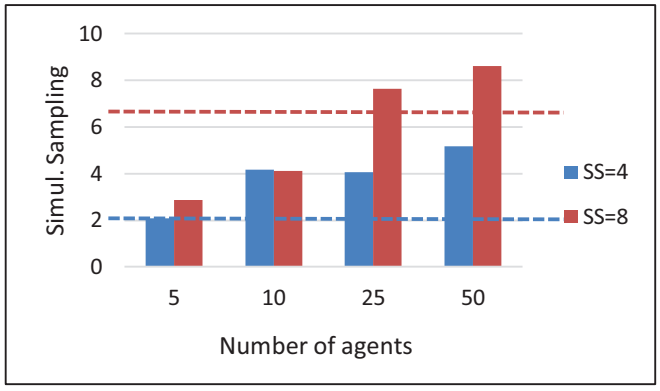

Fig. 6. Simultaneous Sampling: Solution optimized to reach a target simultaneous sampling. As seen, too few or too many agents can make reaching the goal not possible or too easy.

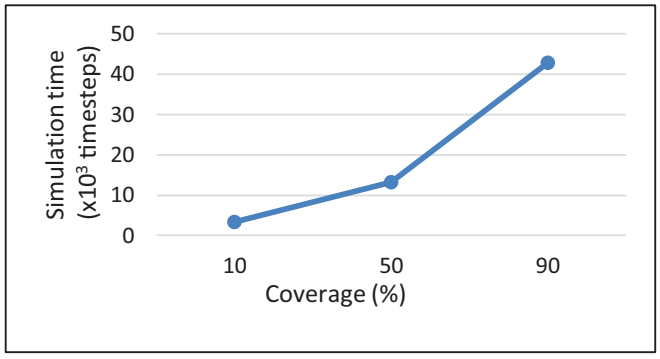

Fig. 7. Coverage: Solution optimized to different coverage levels. The time taken may change non-linearly due to the complexity of the environment.

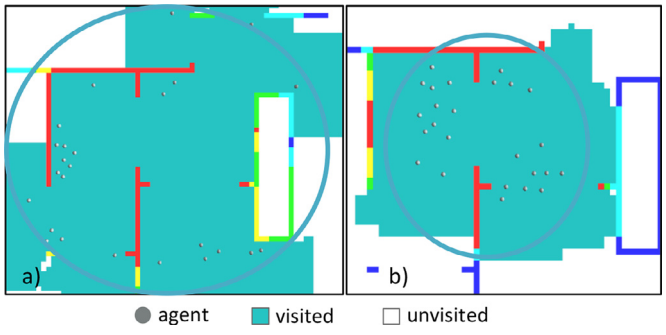

Fig. 8. *Grouping.* We compare a solution optimized for a low grouping indicator value (a) to one of high grouping (b). Both results use 25 agents and are shown at same time step.

Note that while our figures may show the floorplan for an example, the floorplan is in fact not known to the swarming agents. However, for coverage we do use the outer rectangle of the floorplan as the boundaries of a rectangular grid cell array (as discussed in Section 4, using an over-fitted grid works but the value of the coverage indicator does not precisely correspond to actual coverage values).

6.1. Indicators

Figs. 5–7 show the effect of altering indicator values. Fig 5 shows results of using 1, 3, 5, 10, and 25 agents to capture the environment in Fig. 1 with a simultaneous sampling of one (i.e., at least every wall/object is seen once). As expected, a greater number of agents produce a faster acquisition.

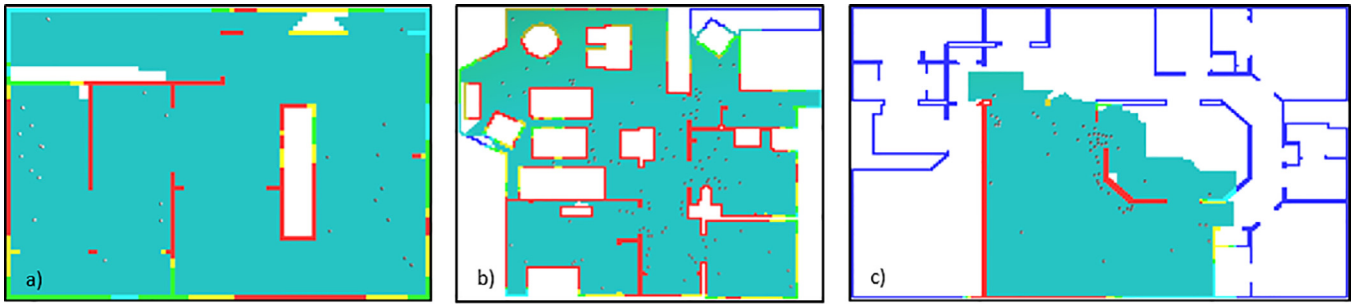

Fig. 9. Swarming Diverse Architecture Spaces: (a) Environment used for simultaneous sampling (SS) and coverage results (shown near completion for SS=4 and 25 agents). (b) Environment with various furniture pieces (shown with 100 agents optimized for maximum coverage in minimum time). (c) Environment with small rooms (shown with 50 agents optimized for high grouping).

Table 1
Fault tolerance. Our model supports agents randomly dying, even at high percentages. These are shown for 25 agents with a desired surface sampling (SS) of four.

Death Rate (%)	Sim. time	SS	Coverage (%)
0	1704	4.07	100
10	1741	3.9	100
20	1814	3.63	100
50	4871	3.55	100

Table 2
Comparison: We compare our approach (bottom) to original boids (did not complete even after 50,000 steps), an optimized version of boids (using components of our solution), and a global sharing method. All results use 10 agents with a goal simultaneous sampling (SS) of 4.

Method	Sim. time (x103 steps)	SS	Coverage (%)
Original Boids	50*	3.51	45
Opt. Boids	11.193	2.48	100
Global	6.067	4.05	100
Local (Ours)	8.467	4.18	100

Simultaneous sampling control is shown in Fig. 6. We show several optimized solutions computed for different numbers of agents and different simultaneous sampling targets through the environment (Fig. 9a). Note that our optimization determines the swarming model parameters that should yield the desired simultaneous sampling. But, recall the agents do not share maps thus said sampling is not a guaranteed outcome. The balance between the simultaneous sampling demand, the number of agents, and environment complexity affects whether the target simultaneous sampling is not reached, reached, or exceeded. We can vary the coverage indicator in order to permit trading-off performance for acquisition incompleteness. Fig. 7 shows the results for various coverage values of the floorplan in Fig. 9a.

Table 1 shows the robustness of our method when agents die during the acquisition process. Fig. 8 demonstrates usage of the grouping indicator. When desired agents can be designed to stay closer together forming groups which may be beneficial in some cases (*e.g.,* pose estimation algorithms). Fig. 9 shows snapshots during the simulated capture though three different floor plans with 25 agents.

6.2. Comparison

Table 2 shows the comparison of our method to alternative approaches. We compare the efficiency of our method to an approximation of the original Boids model [1]. Our implementation of the Boids model uses our infrastructure with hand-picked velocity weights and no explore velocity. We also compare to an optimized Boids models. We use our optimization to find the most suitable velocity weights but still do not include our explore velocity notion. Finally, we adapted our method to send all captured map information to a centralized server which then coordinates the distribution of goal grid cells for exploring. In this case, our method should ideally achieve a similar level of efficiency but does so without requiring a centralized server. As compared to these methods, our approach performs well.

Amongst all our demonstrated results, the velocity weight values determined by our optimization vary significantly *e.g.,* k_e, k_a, k_c, and k_s vary by up to 59%, 83%, 35%, and 83%, respectively. However, the weights tend to not vary much between similar floor plans optimized with the same indicator objectives. Hence, we compute the solution for a floor plan and then use those weights to capture unknown, but expected to be similar, environments.

6.3. Limitations

Our method has a few limitations. First, our method assumes the same swarming behavior is adequate during all stages of capture. Second, our method may not have an advantage for rapidly exploring environments with big open spaces, as a simpler method of dispersing in different directions may provide good results. Third, as mentioned previously, our method cannot guarantee a prescribed simultaneous sampling goal or coverage value is attained, but our simulated results show at least values similar to the desired ones are obtained.

7. Conclusions and future work

We have shown our output-driven swarming approach to environment exploration. Our parameterized swarming model is controlled by an optimization that discovers the best parameter values to yield a desired swarming behavior. Subsequently, our swarming model can be used to sample an architectural indoor environment without requiring a centralized server or map sharing – this is conducive to high scalability. Further, our optimization model defines and uses a set of output-based indicators that enable the operator to predetermine how the environment will be acquired.

As future work, we see several avenues. First, our method currently assumes an external position and orientation (i.e., pose) estimation per agent. We would like to incorporate relative pose estimation by using nearby agents and perhaps additional sensors. Second, we would like to train a neural network with the results of our system so as to permit very fast estimation of behavior parameters and could, in theory, optimize behavior as the agents are discovering a scene. Third, we would like to apply our swarming model to real-world agents and reconstruction, as well perform additional analysis.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was funded in part by National Science Foundation grant 10001387, *Functional Proceduralization of 3D Geometric Models* and National Science Foundation grant 1835739, *U-Cube: A Cyberinfrastructure for Unified and Ubiquitous Urban Canopy Parameterization*.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.cag.2020.02.003.

References

[1] Reynolds CW. Flocks, herds and schools: a distributed behavioral model, 21. ACM; 1987.
[2] Wang X, Ren J, Jin X, Manocha D. Bswarm: biologically-plausible dynamics model of insect swarms. In: Proceedings of the 14th ACM SIGGRAPH/Eurographics symposium on computer animation. ACM; 2015. p. 111–18.
[3] Vicsek T, Czirók A, Ben-Jacob E, Cohen I, Shochet O. Novel type of phase transition in a system of self-driven particles. Phys Rev Lett 1995;75(6):1226.
[4] Hartman C, Benes B. Autonomous BOIDS. Comput Animat Virtual Worlds 2006;17(3–4):199–206. https://doi.org/10.1002/cav.v17:3/4.
[5] Correll N, Hamann H. Probabilistic modeling of swarming systems. In: Springer handbook of computational intelligence. Springer; 2015. p. 1423–32.
[6] Simmons R, Apfelbaum D, Burgard W, Fox D, Moors M, Thrun S, et al. Coordination for multi-robot exploration and mapping. In: Proceedings of the AAAI/IAAI; 2000. p. 852–8.
[7] Scaramuzza D, Achtelik MC, Doitsidis L, Friedrich F, Kosmatopoulos E, Martinelli A, et al. Vision-controlled micro flying robots: from system design to autonomous navigation and mapping in gps-denied environments. IEEE Robot Autom Mag 2014;21(3):26–40.
[8] Beck A, Teboulle M. Fast gradient-based algorithms for constrained total variation image denoising and deblurring problems. IEEE Trans Image Process 2009;18(11):2419–34.
[9] Šťava O, Pirk S, Kratt J, Chen B, Měch R, Deussen O, et al. Inverse procedural modelling of trees. Comput Graph Forum 2014;33(6):118–31. doi:10.1111/cgf.12282.
[10] Weissenberg J, Riemenschneider H, Prasad M, Van Gool L. Is there a procedural logic to architecture?. In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2013. p. 185–92.
[11] Šťava O, Benes B, Měch R, Aliaga DG, Krištof P. Inverse procedural modeling by automatic generation of l-systems. Comput Graph Forum 2010;29(2):665–74. doi:10.1111/j.1467-8659.2009.01636.x.
[12] Vanegas CA, Garcia-Dorado I, Aliaga DG, Benes B, Waddell P. Inverse design of urban procedural models. ACM Trans Grap (TOG) 2012;31(6):168.
[13] Nishida G, Garcia-Dorado I, Aliaga DG, Benes B, Bousseau A. Interactive sketching of urban procedural models. ACM Trans Graph (TOG) 2016;35(4):130.
[14] Mathew CDT, Knob PR, Musse SR, Aliaga DG. Urban walkability design using virtual population simulation. In: Proceedings of the computer graphics forum, 38. Wiley Online Library; 2019. p. 455–69.
[15] Seitz SM, Curless B, Diebel J, Scharstein D, Szeliski R. A comparison and evaluation of multi-view stereo reconstruction algorithms. In: Proceedings of the 2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06), 1. IEEE; 2006. p. 519–28.
[16] Atanasov N, Le Ny J, Daniilidis K, Pappas GJ. Decentralized active information acquisition: theory and application to multi-robot slam. In: Proceedings of the 2015 IEEE international conference on robotics and automation (ICRA). IEEE; 2015. p. 4775–82.
[17] Indelman V, Nelson E, Michael N, Dellaert F. Multi-robot pose graph localization and data association from unknown initial relative poses via expectation maximization. In: Proceedings of the 2014 IEEE international conference on robotics and automation (ICRA). IEEE; 2014. p. 593–600.
[18] Fox D, Ko J, Konolige K, Limketkai B, Schulz D, Stewart B. Distributed multi-robot exploration and mapping. Proc IEEE 2006;94(7):1325–39.
[19] Berman S, Kumar V, Nagpal R. Design of control policies for spatially inhomogeneous robot swarms with application to commercial pollination. In: Proceedings of the 2011 IEEE international conference on robotics and automation. IEEE; 2011. p. 378–85.
[20] Dorigo M, Floreano D, Gambardella LM, Mondada F, Nolfi S, Baaboura T, et al. Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. IEEE Robot Autom Mag 2013;20(4):60–71.
[21] Kushleyev A, Mellinger D, Powers C, Kumar V. Towards a swarm of agile micro quadrotors. Autom Robot 2013;35(4):287–300.
[22] Vásárhelyi G, Virágh C, Somorjai G, Tarcai N, Szörényi T, Nepusz T, et al. Outdoor flocking and formation flight with autonomous aerial robots. In: Proceedings of the 2014 IEEE/RSJ international conference on intelligent robots and systems. IEEE; 2014. p. 3866–73.
[23] Rubenstein M, Cornejo A, Nagpal R. Programmable self-assembly in a thousand-robot swarm. Science 2014;345(6198):795–9.
[24] Cucu L, Rubenstein M, Nagpal R. Towards self-assembled structures with mobile climbing robots. In: Proceedings of the 2015 IEEE international conference on robotics and automation (ICRA). IEEE; 2015. p. 1955–61.
[25] Dirafzoon A, Lobaton E. Topological mapping of unknown environments using an unlocalized robotic swarm. In: Proceedings of the 2013 IEEE/RSJ international conference on intelligent robots and systems. IEEE; 2013. p. 5545–51.
[26] Wagner G, Choset H. Gaussian reconstruction of swarm behavior from partial data. In: Proceedings of the 2015 IEEE international conference on robotics and automation (ICRA). IEEE; 2015. p. 5864–70.
[27] Saska M, Vonásek V, Chudoba J, Thomas J, Loianno G, Kumar V. Swarm distribution and deployment for cooperative surveillance by micro-aerial vehicles. J Intell Robot Syst 2016;84(1–4):469–92.
[28] Lee SK, Becker A, Fekete SP, KrÃ¶ller A, McLurkin J. Exploration via structured triangulation by a multi-robot system with bearing-only low-resolution sensors. In: Proceedings of the 2014 IEEE international conference on robotics and automation (ICRA); 2014. p. 2150–7. doi:10.1109/ICRA.2014.6907155.
[29] Mahadev A, Krupke D, Fekete SP, Becker AT. Mapping and coverage with a particle swarm controlled by uniform inputs. In: Proceedings of the 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS); 2017. p. 1097–104. doi:10.1109/IROS.2017.8202280.
[30] Ramachandran RK, Elamvazhuthi K, Berman S. An optimal control approach to mapping GPS-denied environments using a stochastic robotic swarm. In: Robotics Research. Springer; 2018. p. 477–93.
[31] Reynolds CW. Steering behaviors for autonomous characters. In: Proceedings of the game developers conference, 1999. Citeseer; 1999. p. 763–82.
[32] Gilks WR, Richardson S, Spiegelhalter D. Markov chain Monte Carlo in practice. Chapman and Hall/CRC; 1995.
[33] Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E. Equation of state calculations by fast computing machines. J Chem Phys 1953;21(6):1087–92.
[34] Hastings WK. Monte Carlo sampling methods using Markov chains and their applications. Oxford University Press; 1970.
[35] Harabor DD, Grastien A. Improving jump point search. In: Proceedings of the 24th international conference on automated planning and scheduling; 2014.