Model mining in sensor data for rapid terrain analysis

Frido Kuijper, Ewan Demeur, Remco van der Meer, Vera Bekkers, Ruben Smelik
TNO Defence, Safety and Security Research Institute
The Hague, The Netherlands
{frido.kuijper | ewan.demeur | remco.vandermeer | vera.bekkers | ruben.smelik}@tno.nl

ABSTRACT

Military simulation applications put strong requirements on terrain modelling. Large mission and training areas need to be represented in detail, while users expect these models to be available at ever shorter lead times. Capabilities are needed that rapidly transform sensor data into models that fully represent the complexity of the mission environment. Industry seems to have solved part of the problem with mature photogrammetric techniques and LiDAR data acquisition. However, the data delivered by these techniques is often limited to a geometric and only visual model that has little semantics and as such is not ready for simulation.
Current sensor data analysis techniques result in labelled imagery and point clouds, assigning semantics to pixels and points. At best, the points are then converted into semantic polygons or line segments. The challenge is to find complete models that match with the geometry and semantics of the points. The research presented in this paper addresses this challenge. We seek techniques that directly extract semantically rich and simulation-ready models from sensor data. Our hypothesis is that procedural modelling techniques are key to the solution and that innovative application of modern data analysis techniques is required to delve instances of these procedural models from the sensor data. We introduce the concept of model mining to refer to the process that finds these models by fitting optimized models to sensor data.
In our paper we report on results we have achieved with model mining applied to a drone based point cloud data set. We studied particle swarming and genetic optimization techniques to find procedural models within the data. Model mining is a complex problem that needs extensive research to mature. We hope this paper will motivate others to take on the challenge of model mining and bring rapid terrain analysis to the military simulation community.

ABOUT THE AUTHORS

Frido Kuijper is a senior research scientist at TNO in the Netherlands. With a background in computer science (MSc degree at Delft University of Technology) he has been working in the field of military modelling and simulation for thirty years now. His research team has a focus on the modelling of military mission and training areas, making detailed and up-to-date virtual environments available to simulation applications. He is the program lead for a research program that seeks to extend the knowledge on virtual mission environments that comply with the complexity and dynamics of current and future military conflicts.

Ewan Demeur is a research scientist at TNO in the Netherlands. Ewan has been researching the various optimization techniques that were subject in this study. He holds a MSc degree in Artificial Intelligence from Maastricht University.

Remco van der Meer is a research scientist at TNO in the Netherlands. Remco has been contributing to various parts of our virtual mission environment development suite, amongst which the techniques addressed in this study. He holds a MSc degree in Mathematics and a MSc degree in Computer Science from Delft University of Technology.

Vera Bekkers is a research scientist at TNO in the Netherlands. Vera is currently focusing on vegetation analysis and modelling. She holds a MSc degree in Geo-Information Science from Wageningen University.

Ruben Smelik is a senior research scientist at TNO in the Netherlands. He has been in the field of modelling and simulation for military applications since 2007. He holds a MSc degree in computer science from Twente University and earned a PhD degree from Delft University of Technology based on his thesis on the automatic creation of 3D virtual worlds.

Model mining in sensor data for rapid terrain analysis

Frido Kuijper, Ewan Demeur, Remco van der Meer, Vera Bekkers, Ruben Smelik
TNO Defence, Safety and Security Research Institute
The Hague, The Netherlands
{frido.kuijper | ewan.demeur | remco.vandermeer | vera.bekkers | ruben.smelik}@tno.nl

INTRODUCTION

Military simulation applications require terrain with functionality, not just looks

In our vision, the military user of simulation technology should be served with a capability that always offers an up-to-date terrain model of tomorrow's operation. Based upon as recent as possible sensor data from the scene of operation, be it a mission area or some training area, a virtual representation of the area needs to be generated quickly and accurately. With current availability of sensor data, like satellite imagery, detailed aerial imagery and accurate lidar point measurements, a quick wrap-up of that data into a simulation model seems like a cracked case. But reality is that it is not that simple. Military simulations put strong requirements on the terrain model's properties and quality. Industry has well succeeded in generating high quality mesh models (e.g. Figure 1.a) from large scale sensor data, but these models are mainly providing 'the looks' and lack the properties that enable simulation of, e.g., manoeuvre (both outdoor and indoor), physical interactions, weapon impact and Computer Generated Forces behaviour (all available in Figure 1.b).

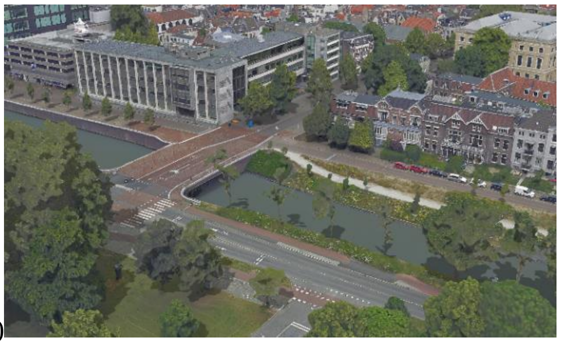
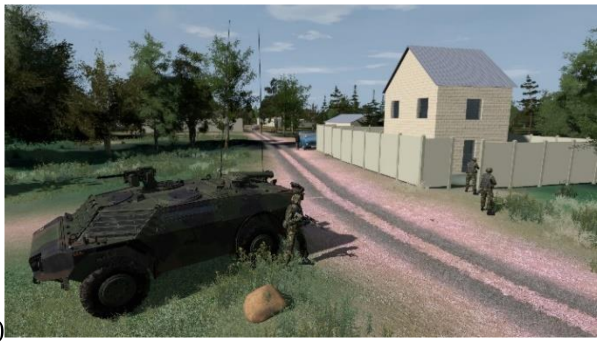
(a)  (b) 

Figure 1. Creation of high quality mesh models from sensor data is considered solved by industry (example (a) 3D Utrecht by ESRI Netherlands and Kavel 10), but these models lack the functionality as required by military simulation applications (sample (b) virtual training area in VBS by TNO).

Our research hypothesis: rapid terrain analysis starts with a (procedural) model

In order to obtain terrain models that fulfil all requirements of military simulation, our hypothesis is that rapid (i.e., automatic) terrain analysis techniques should start at the bottom line: the model that we are looking for. In essence, automatic terrain analysis is a search problem, that shall find models fitting the sensor data. We believe that the solution space for this search problem is to be described as a set of *procedural* models.

Let us illustrate this hypothesis with the case of building models. The building model as shown in Figure 1.b was produced by using an interactive modelling tool, UrbanBuilder by RE-liON (see Figure 2). The building model is constructed by drawing and specifying a set of semantically rich floorplans (Kuijper, 2019). The key properties of the building are all included in the specifications: geometry of the walls, doors, windows, but also the materials that they are composed of and elements like stairways that connect the floors. From this specification, all that is required by the simulation target systems can be generated by computation. The resulting model in Bohemia's Virtual Battle Space (VBS) is fully functional: human controlled entities can navigate the building, but also artificial intelligence controlled entities are able to find their ways, fire effects can take building materials into account and damage models can be (pre)computed as needed. The resulting building model is a sample of a *procedural* model: from a set of specifications (the floorplans) and a set of procedures to transform these specifications, a simulation-ready model can automatically be derived.

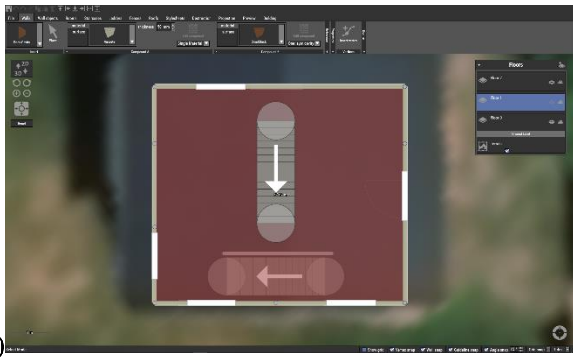
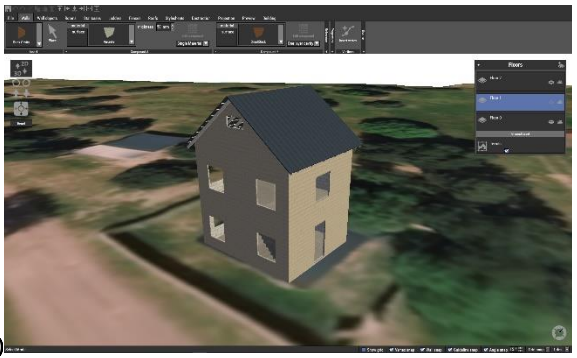
(a)  (b) 

Figure 2. A procedural building model that is generated from a set of semantically rich floor plan descriptions, created with RE-liON UrbanBuilder tool. The (b) 3D model is automatically computed from the (a) floorplans drafted in 2D.

Procedural models can be very powerful. It takes considerable effort to develop a set of specifications and the algorithms to transform these into simulation-ready models, but once available they are powerful enough to describe complex military mission and training areas with relatively little effort (compared to fully manual modelling using a standard 3D modelling tool). The intrinsic completeness of the model, i.e., all semantics are fully known, even allows for automatic derivation of dynamic effects on the model such as damage (Smelik et al, 2019). Various tools are available to implement procedural models. A well-known tool is CityEngine (Kelly, 2021) that is based on a shape grammar system originally developed by (Müller et al, 2006). Apart from buildings, procedural modelling can essentially be applied to any part of the terrain model. Figure 3 shows an example of a bridge model that is procedurally generated from a given terrain elevation model and linear vector describing the location of the bridge feature. The model was generated using Houdini SideFX, another well-known tool for procedural models.

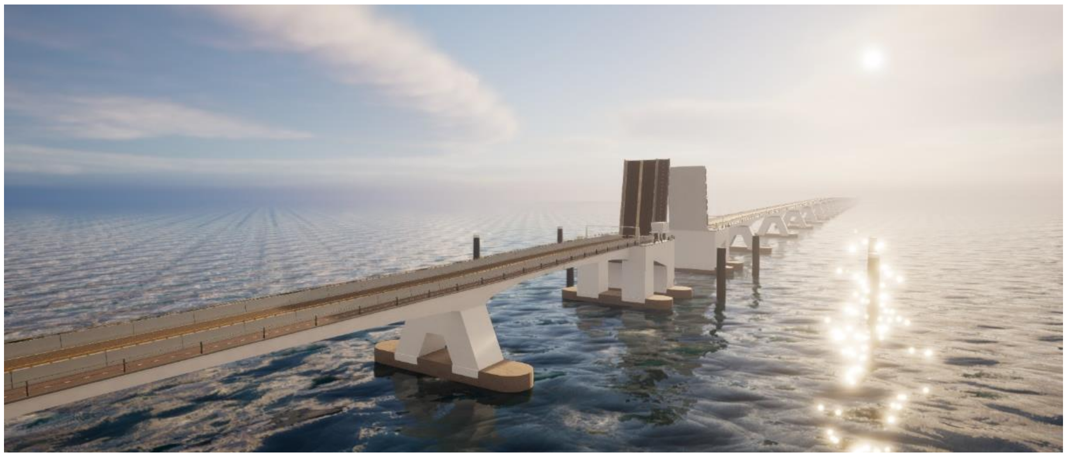


Figure 3. A procedural bridge model, automatically generated from terrain elevation data and a linear vector describing the location and configuration of the bridge. Generated with Houdini, exported to and shown here in Unreal Engine.

The challenge: model mining in sensor data

Now that we have defined the class of procedural models as a suitable method to describe terrain in virtual mission and training areas, the key challenge of rapid terrain analysis can be defined as what we refer to as *model mining*. Model mining is the process that searches for procedural model instances that completely and accurately describe a terrain, based upon available sensor data. We introduce this term to emphasize the fact that we look for *models* rather than just pixel labels or vector features indicating the presence of some terrain object, while the *mining* part refers to the complexity of the process, prompted by the high dimensionality and size of both the input data space as well as the solution space.

RESEARCH OBJECTIVES

Our research program seeks complete end-to-end solutions that operate on multiple types of sensor data. Within that overall scope, this paper has a more detailed scope, digging into just one piece of the puzzle. It focuses on model fitting techniques that can be used to implement the model mining concept that we envision. More specifically, we studied two model fitting techniques, seeking to answer the following questions:

- How can *particle swarming optimization* be used to fit procedural models on point cloud sensor data?
- How can a *genetic algorithm* be used to fit procedural models on point cloud sensor data?
- How do these techniques compare to each other and to other approaches?

Notice that these questions do not address the current popular approach of using complex deep learning models for data science challenges like the model mining challenge that we took on. This is a deliberate choice, as we choose to first study techniques that do not rely on extensive model training and labelled datasets, but can be applied in an unsupervised way.

RELATED WORK

Unlike the well-known and similar terms such as *data mining* and *crypto mining*, our self-introduced term *model mining* is not a great keyword when searching for related work. The question is, if related work is available that uses a model fitting approach that starts with semantic rich and simulation-ready (procedural) models. Recent work presented at I/ITSEC includes (Usumezbas et al, 2020), in which point cloud data and aerial imagery are attributed with semantic information. Their work flow results in a mesh model that is semantically attributed, and as such is a good starting point to mine models. However, the approach does not include a model fitting strategy, nor does it result in complete simulation ready models for each individual object in the terrain. In (Chen et al, 2019) an automated work flow to extract object information from photogrammetric point clouds is presented. They use a deep learning model to classify the points into man-made structures, vegetation and ground. The vegetation points are fit to a set of simulation-ready tree models. However, they conclude that a model fitting strategy on photogrammetric point clouds is not suitable, given the lack of geometric information on the lower and inner parts of the trees.
Many publications can be found on automatic model generation for buildings. Most of them focus on geometric reconstruction methods. In (Li and Wu, 2020), a method is described that is able to generate a topologically correct geometric model of complex buildings as found in dense urban environments, but without any semantic attribution. In (Zhang, Li and Shan, 2021) an approach to building reconstruction is described that is similar to the work we present in this paper, as it optimizes parametric models. Their experiments work on isolated point clouds of buildings, based on known footprints. PointNet++ is used to predict the building construction type and select a parametric building shape, which is subsequently fit to the data using RANSAC (to detect roof ridges) and a least squares optimization technique.
Impressive work on building reconstruction was done by Blackshark.ai (Hollosi et al, 2022). They developed a fully automated work flow to generate 3D building models from satellite and aerial imagery, making use of deep learning techniques. Likewise as promoted in this paper, they use procedural modelling techniques to represent the buildings and expand them as required at the simulation client side. Their system does rely on an extensive training process for the deep learning models being used.
Related work using model fitting approaches can also be found on the subject of bridge models. Bridge modelling has historically more often been approached using procedural modelling techniques. The reason is obviously in the fact that bridges are less complex in terms of architectural variety than buildings. Good results have been achieved by (Goebbels, 2021), (Goebbels and Dalitz, 2021). They use point clouds as input data, but also cadastral footprint data that represents the bridge location, thus simplifying the fitting problem. Their method is not relying on procedural bridge models (with predefined bridge types, similar to primitive building types) to be fit to the data, but rather reconstructs individual bridge elements as observed in the point cloud data.

METHOD

To address the research objectives of this paper, we implemented two algorithms (a particle swarming optimization algorithm and a genetic algorithm) to fit procedural models (for a light pole object and for a building object) within point cloud data and experimented with them using a given test data set, as further detailed in this section.

Test data set

The test data used in this paper is a point cloud data set (see Figure 4) that is derived from drone footage of the Oostdorp military training village, located on the Dutch Army's training area Harskamp. The data was acquired by the Dutch Defence Geographic Agency using a Trinity F90+ drone, carrying a Sony RX1 RII sensor payload (nadir oriented). Images were captured with a ground sampling distance of approximately 2 cm. The photogrammetrically derived point cloud has 585,805,970 points (approximately 1,500 points per square meter) and is attributed with RGB colour information.

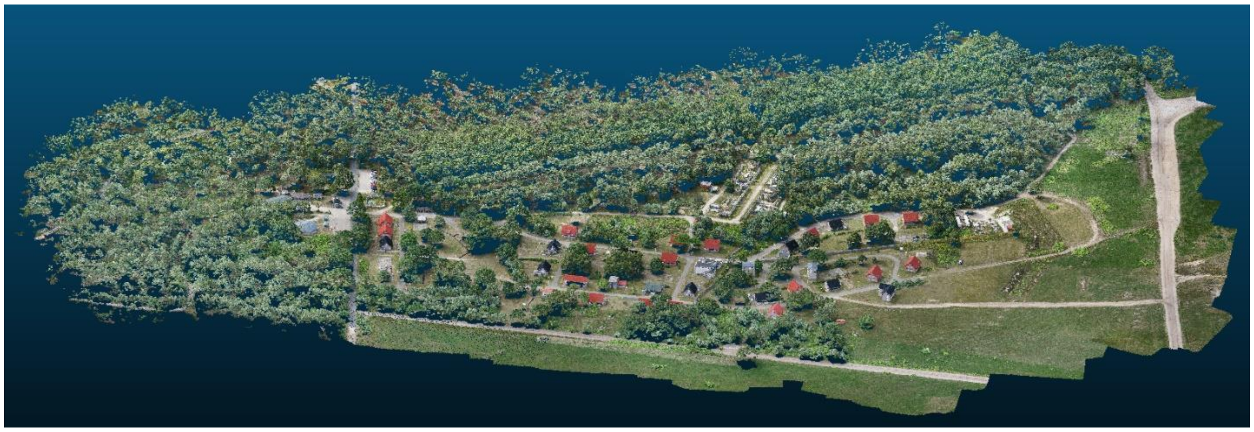


Figure 4. The test data set: a photogrammetrically derived high resolution point cloud of a military training village.

From the test data, we derived four smaller test point clouds to experiment upon, as illustrated in Figure 5. The first two test samples (POLE1, POLE2, consisting of 5,350 and 10,401 points respectively) target at detecting a simple object: a light pole. Notice that the light poles are both only partly covered by the point cloud data. In the first sample, it seems hard to find a light pole, but actually it is not that hard since there is no ground clutter in this sample – just a (small) piece of light pole on top of the ground and a light box up in the air. In the second sample, the situation is complicated by the presence of tree leaves above the light pole and some clutter on the ground. The second pair of samples (HOUSE1 and HOUSE2, consisting of 1,423,249 and 1,604,646 points respectively) cover a somewhat larger area and include a building model as well as other objects such as light poles, vegetation and other clutter objects like the car parked in front of the building.

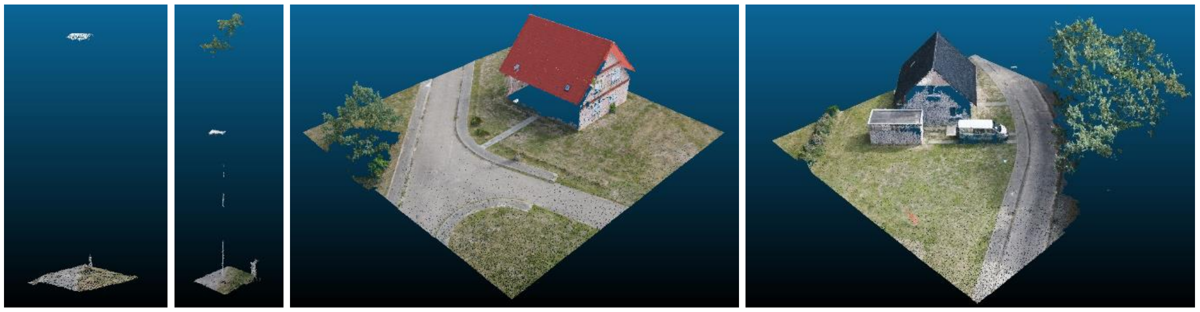
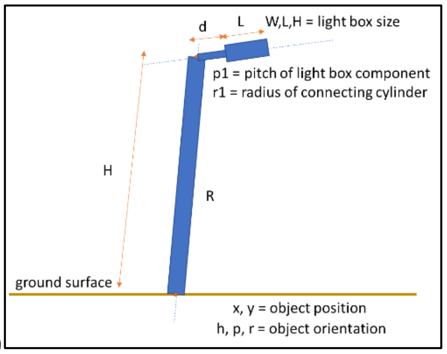


Figure 5. Four samples from the data set, that are used for experimentation. From left to right the samples are identified as POLE1, POLE2, HOUSE1, HOUSE2 in this paper.

Models to fit

As the focus of this study is on the model fitting techniques, we limit ourselves to the fitting of two simple types of parametric models: light poles and hip roof buildings (with gable roof buildings included in that building type when the hip spans the complete building), see Figure 6. Though simple, the models illustrate the idea of mining procedural models from sensor data: once the parameters are known, the models can be procedurally expanded to simulation ready objects in a terrain representation.

(a) 
d
L
W,L,H = light box size
p1 = pitch of light box component
r1 = radius of connecting cylinder
H
R
ground surface
x, y = object position
h, p, r = object orientation

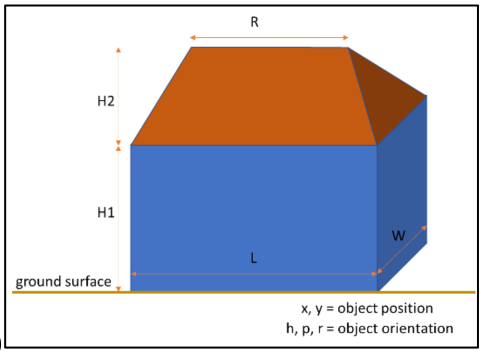
(b) 

Figure 6. The model types we fit in this study: (a) a light pole model with 13 parameters and (b) a building model with 10 parameters.

The task of the model mining technique is to find a model instance that best fits the sensor data. For the light pole model, this means finding a set of 13 parameters, while for the building model, 10 parameters are to be optimized. Notice that the z-coordinate of the object position is considered to be taken from a ground surface data source.

Cost function design

Both techniques – particle swarming and genetic algorithm – need a cost function that measures the quality of a possible solution to the fitting problem. In our study, we rely on the geometric fitness of the solution only, i.e., we measure how well the *shape* of the proposed model solution fits the points in the point cloud. The cost C of a model instance defined by its parameters $\overline{m}$ (as illustrated in Figure 6) given point cloud P is defined in equation 1.

$$\mathrm{C}(\overline{m}, P) = \sum_{p \in P} f_\lambda(|p - M(\overline{m})|) \qquad (1)$$

Where $|p - M(\overline{m})|$ defines the minimum Euclidian distance between point p and the geometry of the procedural model M given parameters $\overline{m}$. This distance is handled as a signed distance, defined to be negative when point p is considered *inside* the models geometry and positive otherwise. The key of the cost function is in the shaping that function f applies to the distance between a point and the model, for which we chose the function as defined in equation 2. Other than a more straight forward least squares cost function, equation 2 has as desirable characteristic that only points that are in the vicinity of the model have discriminating influence on the cost of a solution.

$$f_\lambda(d) = 1 - \frac{\lambda}{\lambda + d^2} \qquad (2)$$

The resulting cost function per point is depicted in Figure 7. As the function is based on signed distance to the model, points inside and outside the model could be penalized with different costs, but our experiments resulted in choosing a symmetrical function that penalizes equally inside and outside the model. The smoothing parameter λ determines how tight the points shall match the model. Smaller values will lead to more accurately fit models, but will hinder the search for global minima.

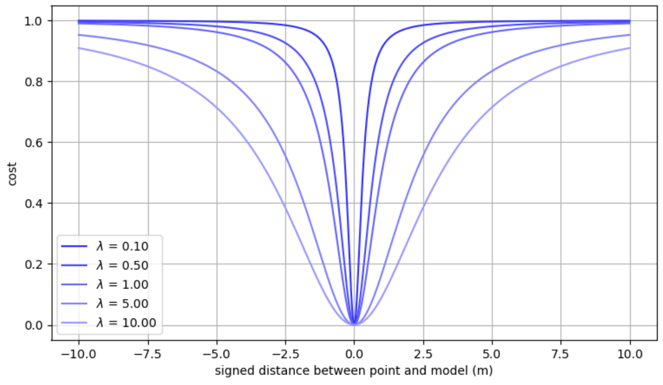


Figure 7. The cost shaping function assigns cost to points that are distant to the model's geometry.

Particle swarming algorithm

Particle swarm optimization (PSO) was introduced by (Kennedy and Eberhart, 1995) as a technique to solve high dimensional, non-linear optimization problems. An interesting review on its applications was written by (Bonyadi and Michalewicz, 2017). PSO can be used to find a model instance $\overline{m}$ for which the cost function $\mathrm{C}(\overline{m}, P)$ is minimized, i.e., the model's geometry $M(\overline{m})$ fits the point cloud P. This is done by creating a set of N particles, each representing an instance of the procedural model, defined by its parameters $\overline{m}_i$. The particles are instantiated randomly and will then move around the multidimensional parameter space in search for the minimized cost state. Each iteration, the particles update their state $\overline{m}_i$ and associated cost $\mathrm{C}(\overline{m}_i, P)$. The basic idea of PSO is that each particle will mimic aspects of social behaviour: it can move *individually* towards its own cognitive idea of what is best (explorative behaviour) and it can move *socially* towards what is globally considered best by the swarm of particles (converging behaviour). PSO has several parameters - coefficients that control the individual versus social behaviour and an inertia coefficient that controls velocity - to control the swarm's behaviour. Several extensions were proposed to adaptively control those parameters, such as the adaptive inertia control proposed in (Xu et al, 2021). We chose to implement the adaptive particle swarm optimization (APSO) algorithm proposed by (Zhan et al, 2009). APSO needs no manual tuning of the parameters that control particle behaviour, and it is more efficiently converging to the best solution. To control the swarm's behaviour, APSO adaptively chooses at each iteration between four states of behaviour, defined as *exploration*, *exploitation* (exploration around local minima), *convergence* and *jumping out* (trying to find a new, better global best when converged to some local minimum) each steering the behaviour control parameters towards values that support the intent of the state. The algorithm is described in detail in (Zhan et al, 2009).

Genetic algorithm

The second model fitting technique we experimented with is a genetic algorithm. Genetic algorithms have been around for a long time, and can be applied to the high dimensional search problems inherent to our model mining challenge. For an overview of using genetic algorithms in search problems, see (Goldberg, 1989). Where PSO uses a swarm of N particles, the genetic approach uses a population of N individuals, where each individual as in PSO defines a model instance defined by its parameters $\overline{m}_i$. Both algorithms use the same cost function $\mathrm{C}(\overline{m}, P)$ to find the best model, the one with the minimum cost. The genetic approach is to step through a process of *selection* (of the individuals from the population that we want to generate offspring), *crossover* (how children inherit the genes of their parents) and *mutation* (how children get random changes in their inherited genes) to enhance the population by *inserting* new offspring to replace existing individuals on every iteration. In our implementation we used the following strategies: roulette wheel selection, single point crossover, scramble mutation and random reinsertion of children, see (Katoch et al, 2021).

Data pre-processing

There is two sorts of pre-processing applied to the input point cloud. Firstly, the point clouds are down sampled to a lower resolution. The resolution depends on the type of model that is matched: the light poles need a higher point density than the buildings. The down sampling has little impact on the solutions found, and in return gives considerable performance gain. Secondly, we filter ground points based upon the given ground surface elevation data.

Implementation and test hardware

All results presented in this paper are based on an implementation of the algorithms in C++, using CUDA to accelerate the cost function evaluation, the particle velocity and position update, and the genetic offspring generation (crossover, mutation and reinsertion). The experiments were executed on a Windows 10 workstation equipped with an AMD Threadripper PRO 5975WX CPU with 256 GB RAM and an NVIDIA RTX3090 with 24 GB RAM.

RESULTS AND DISCUSSION

Model solutions found by the algorithms

The first, qualitative, question that is asked when running the model fitting algorithms on our test point clouds is: does the algorithm find the best model solution? Figure 8 illustrates how the APSO algorithm succeeds to converge to the best light pole model that can be found in the POLE1 point cloud. Table 1 provides an overview of all configurations that were tested and the outcomes. Observe that not all models have been fit correctly. Figure 9 visually illustrates the solutions that were correctly found. The small light pole tests need approximately 200 iterations to converge, while the building solutions are found (both for APSO and GA) within 1000 iterations.

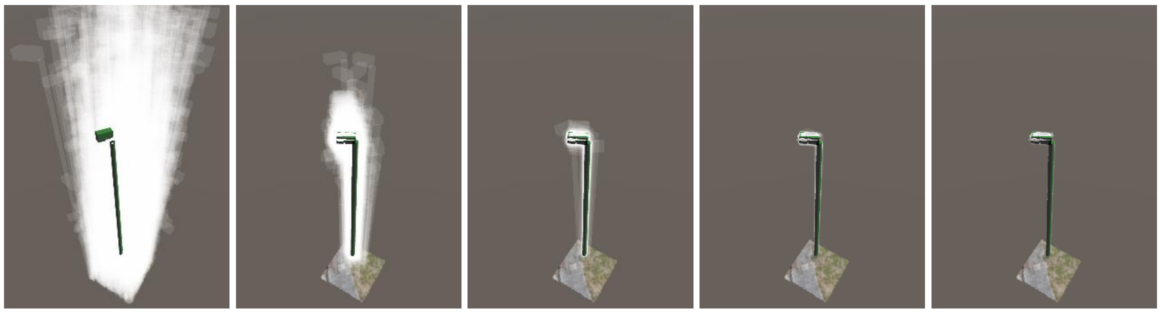


Figure 8. A visualization of how APSO finds a solution for the light pole model in POLE1, from the first iteration (left), to the state where all particles converged (right). Iterations 0, 10, 50, 100 and 200 are shown. In white a subset of the individual particles, in green the current best solution.

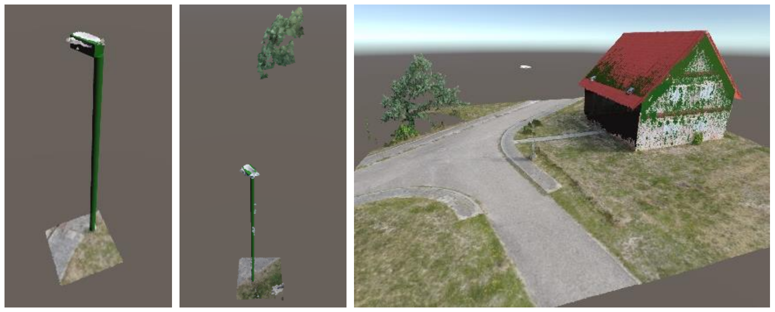


Figure 9. Correct model solutions found on the test data (a) POLE1, (b) POLE2 and (c) HOUSE1.

Table 1. Overview of the results on the various input points clouds and model types using the two algorithms.

Model type	Algorithm	Point cloud	Resample resolution (m)	λ of cost function	Swarm / Population size (#)	Correct solution found?	Time per # per iteration (µs)
light pole	APSO	POLE1	0.05	1.0	15,000	YES	1.760
light pole	APSO	POLE2	0.05	1.0	15,000	YES	1.820
light pole	APSO	HOUSE1	0.05	1.0	15,000	NO	8.409
light pole	APSO	HOUSE2	0.05	1.0	15,000	NO	12.806
building	APSO	HOUSE1	0.50	5.0	15,000	YES	3.660
building	APSO	HOUSE2	0.50	5.0	15,000	NO	4.662
light pole	GA	POLE1	0.05	1.0	2,500	YES	19.539
light pole	GA	POLE2	0.05	1.0	2,500	YES	20.176
light pole	GA	HOUSE1	0.05	1.0	2,500	NO	28.667
light pole	GA	HOUSE2	0.05	1.0	2,500	NO	28.892
building	GA	HOUSE1	0.50	10.0	2,500	YES	31.240
building	GA	HOUSE2	0.50	10.0	2,500	NO	31.396

On incorrect solutions and cost function shape

In a way, our results could be judged as disappointing: half of the cases in Table 1 report to not have found a correct solution. On the other hand, the fact that we have both cases that positively and robustly find correct model solutions, as well as cases that fail consistently, offers good information to develop our knowledge. In cases when techniques like PSO and GA do not work, the first suspected cause is the cost function that is used. Also in our case, the most likely cause is in the cost function, supported by the fact that both APSO and GA find the same incorrect solution. Figure 10 shows what incorrect solutions are found as the model with minimum cost.

The incorrect light pole found in the building façade (Figure 10.a) can be explained from the large number of points that are close to the model, many more than there are points around the true light pole position. In fact, the cost of the true light pole within this scene is not the absolute minimum. One of the problems with our current cost function is

that it is not assigning enough cost to the points that are directly outside (or inside) the model. A cost function that would increase that particular cost would better describe the true solutions, however, it will hinder the optimization process as a ‘cost bump’ is created around the true model positions.
The incorrect building that is found in the tree (Figure 10.b) is a different case, in which the cost of the found solution is actually *higher* than the cost of the true solution. But apparently, both APSO and GA are pushed towards an incorrect local minimum.
It shall be noted that the algorithm just tries to minimize a cost function. The resulting minimum cost value is not a *metric* on which can be decided whether a valid model instance has been found.

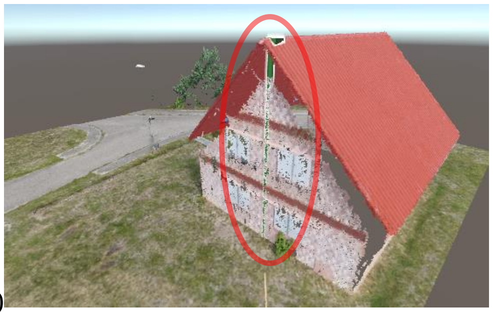
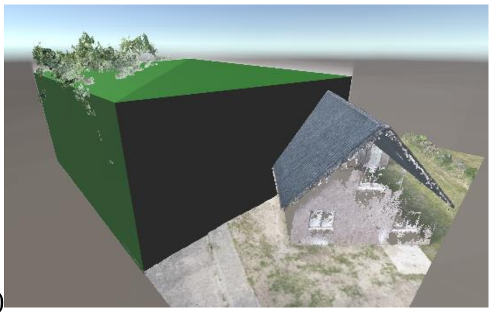
(a)  (b) 

Figure 10. With the current cost function approach, our algorithms (a) incorrectly fit the light pole to a building façade and (b) even the building model in HOUSE2 is incorrectly fit.

On particle swarm optimization versus genetic algorithm
The comparison between the APSO approach and the GA approach is not straight forward on the basis of our current results, and considered undecided for the moment. Both algorithms perform equally in terms of qualitative success/failure to find a correct solution. A final decision between the two (or a decision to discard both) will depend on possibilities to implement improvements and on performance aspects.

On swarm and population size
Table 2 shows the results to determine the minimum required number of particles for APSO and population size for GA. The results show that GA needs considerably less individuals in its population to work effectively. Note that the λ parameter of the cost function was set to the value that worked best for the respective algorithm.

Table 2. Overview of the results on testing different swarm and population sizes for the same model to be fit.

Model type	Algorithm	Point cloud	Resample resolution (m)	λ of cost function	Swarm / Population size (#)	Correct solution found?	Time per # per iteration (µs)
building	APSO	HOUSE1	0.50	5.0	500	NO	8.066
building	APSO	HOUSE1	0.50	5.0	1,000	NO	4.988
building	APSO	HOUSE1	0.50	5.0	5,000	NO	3.942
building	APSO	HOUSE1	0.50	5.0	10,000	NO	3.469
building	APSO	HOUSE1	0.50	5.0	15,000	YES	3.660
building	GA	HOUSE1	0.50	10.0	500	NO	10.468
building	GA	HOUSE1	0.50	10.0	1,000	YES	14.670
building	GA	HOUSE1	0.50	10.0	5,000	YES	55.716
building	GA	HOUSE1	0.50	10.0	10,000	YES	87.014

On performance
Both Table 1 and Table 2 provide insight in the performance of the two algorithms tested. The timings given are the mean time needed per swarm/population member, per iteration. The rationale behind the figures is that both APSO and GA share the efficient and scalable GPU cost computation, but the GA implementation has a far more significant load on the CPU for performing the selection of parents and reinsertion candidates. Hence, APSO has smaller computation times per particle. This advantage of APSO is countered by the effect that GA needs smaller population sizes than the swarm size needed by APSO. When APSO and GA both are set to required sizes, APSO will still have a performance advantage.

CONCLUSIONS AND FURTHER WORK

No success without failure

We have shown how a particle swarming algorithm (APSO) and a genetic algorithm (GA) can be used to perform two relatively simple examples of model mining: finding parametric light pole and building models in a point cloud. The results we present show both success and failure, and do not offer a ready-to-go solution for the rapid terrain analysis challenge. Also, our experiments only touch upon the high complexity and dimensionality of the solution space that has to be searched when model mining for real-life applications. However, the presented results do provide insight into the conditions under which APSO and GA succeed or fail to find fitting models, and also into the performance characteristics when using CUDA-based GPU acceleration.

The prospect in comparison to existing work flows

The conceptual advantages of our approach over commonly used work flows are clear, and worth exploring. We presented fully *unsupervised* automatic model fitting algorithms. Most existing or proposed automatic work flows involve pre-processing steps that depend on, often expensive, supervised training. For point clouds, semantic segmentation (exemplified in Figure 11 for our test data) is often used to aid model fitting. Secondly, although the presented models were relatively simple parametric models, our goal is to target complete and rich procedural models as a direct output of our model mining techniques. Other than geometric reconstruction techniques, these models are semantically rich and detailed, thus enabling the automatic derivation of virtually any required run-time simulation model, including the modelling of dynamic effects.

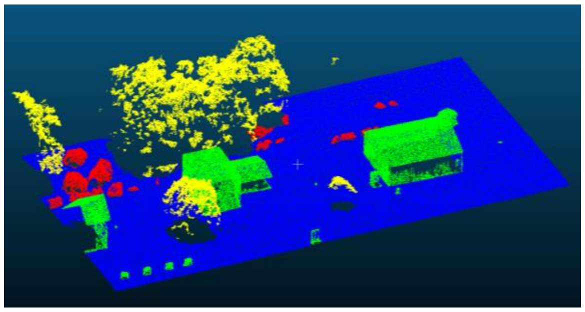


Figure 11. Semantic segmentation of point clouds is often used to simplify the problem of model mining, at the cost of supervised training, something we try to avoid.

Further work

A next step to take on our side is to make the APSO and GA algorithms work reliably in the current test cases. We will revisit the cost function design, and survey alternative strategies to improve robustness. A possible strategy is to divide the solution space in sectors and split the swarm in sub-swarms that are constrained to a certain sector. This strategy is comparable to the GA island model proposed by (Whitley, Rana and Heckendorn, 1998). We will also study the design of metric functions that help discriminate between false and true solutions. After that, we will extend the search space by handling the full test data set and considering the full complexity of all buildings, other man-made objects and vegetation. The mining of different types of models in concert is both a challenge as well as an opportunity. An incorrectly fitted light pole model could for example be overruled by a fitted building model that has a higher quality metric. We might introduce semantic segmentation as a pre-processing step, but will keep striving after unsupervised techniques, or at least avoid techniques that rely on a complex training process.

Our research program will continue the search for methods and techniques that make the model mining concept work. We hope that the presented work will motivate others to join that search.

ACKNOWLEDGEMENTS

This research is part of a four year (2022-2025) research program named *Hybrid and Dynamic Virtual Mission Areas* sponsored by the Dutch Ministry of Defence. The authors would like to thank the Dutch Army Simulation Center, the KIXS knowledge and innovation department as well as the Dutch national Defence Geographic Agency for their support to this research program.

REFERENCES

Bonyadi, M., & Michalewicz, Z. (2017). Particle Swarm Optimization For Single Objective Continuous Space Problems: A Review. *Evolutionary Computation, 25(1), 1-54.*

Chen, M., Feng, A., Mcculough, K.,Prasad, B., Mcalinden, R., Soibelman, L., & Enloe, M. (2019). Fully Automated Photogrammetric Data Segmentation and Object Information Extraction Approach for Creating Simulation Terrain . Paper presented at the *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC).*

Goebbels, S. (2021). 3D Reconstruction of Bridges from Airborne Laser Scanning Data and Cadastral Footprints. *Journal of Geovisualization and Spatial Analysis, 5(1),* 1-15.

Goebbels, S., & Dalitz, C. (2021). Reconstruction of Bridge Superstructures from Airborne Laser Scanning Point Clouds. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 2021(VIII-4),* 121-128.

Goldberg, D. (1989). *Genetic Algorithms in Search,* Reading, MA: Addison-Wesley Publishing Company.

Hollosi, A., Menzel-Berge, T., Walter, H., & Lahm, D. (2022). Automated 3D Building Generation at Global Scale Based on Satellite Imagery. Paper presented at the *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC).*

Katoch, S., Chauhan, S., & Kumar, V. (2021). A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications, 80(5),* 8091-8126.

Kelly, T. (2021). CityEngine: An Introduction to Rule-Based Modeling. *The Urban Book Series, Urban Informatics,* 637-662.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *International Conference on Neural Networks*, 1942-1948.

Kuijper, F. (2018) Building virtual training areas, challenges and solutions. Paper presented at the *International Training Equipment Conference (ITEC).*

Li, Y., & Wu, B. (2020). Automatic 3D reconstruction of complex buildings from incomplete point clouds with topological relation constraints. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 2020(V-5),* 85-92.

Müller, P., Wonka, P., Haegler, S., Ulmer, A., & Gool, L. van (2006). Procedural Modeling of Buildings. *ACM Transactions on Graphics, 25(3),* 614-623.

Smelik, R., Wermeskerken, F. van, Krijnen, R., & Kuijper, F. (2019). Dynamic synthetic environments: a survey. *Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, 16(30),* 255-271.

Srinivas, M., & Patnaik, L. (1994). Adaptive Probabilities of Crossover Genetic in Mutation and Algorithms. *IEEE Transactions on Systems, Man and Cybernetics, 24(4),* 656-667.

Usumezbas, A., Matei, B., Samarasekera, S., & Kumar, R. (2020). Semantics-Aware 3D Segmentation and Modeling System for Immersive Simulations and Training Scenarios. Paper presented at the *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC).*

Whitley, D., Rana, S., & Heckendorn, R. (1998). The Island Model Genetic Algorithm: On Separability, Population Size and Convergence. *Journal of Computing and Information Technology, 7(1).*

Xu, L., Song, B., & Cao, M. (2021). An improved particle swarm optimization algorithm with adaptive weighted delay velocity. *Systems Science and Control Engineering, 9(1),* 188-197.

Zhan, Z., Zang, J., Li, Y., & Chung, H. (2009). Adaptive Particle Swarm Optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 39(6),* 1362-1381.

Zhang, W., Li, Z., and Shan, J. (2021). Optimal Model Fitting for Building Reconstruction from Point Clouds. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 14,* 9636-9650.