Real-time Visualisation of Densely Populated Urban Environments: a Simple and Fast Algorithm for Collision Detection

Franco Tecchia[1]* and Yiorgos Chrysanthou†

*Laboratorio PERCRO,
Scuola Superiore S.Anna,
Pisa, Italy
franco@percro.sssup.it

†Department of Computer Science
University College London
Gower Street
London WC1E 6BT
Y.Chrysanthou@cs.ucl.ac.uk

Abstract
In this paper we present a fast collision detection method with an application to a densely populated urban environment. The method uses graphics rasterization hardware to discritise the environment and to create a look-up table with heights which can be used not only for preventing the humans from walking through buildings but also for determining and adjusting their elevation on the model without having to query the geometrical database.

Keywords: collision detection, crowd simulation, hardware rasterization, virtual cities

1 Introduction

Recently we have seen the appearance of many geometric models of towns and cities which are being used in a variety of applications. These applications range from town planning to acclimatisation using virtual environments, and travel rehearsals. However, these models are hardly ever populated with more than a few individuals.

At UCL we have a polygonal model of London that covers a large portion of the city, 160 km in total, with some parts of it modelled in great detail. This model was originally developed through the COVEN project [COVEN] with the intention to be used in Virtual Reality for travel rehearsals where a person unfamiliar with an area can learn his way to a particular place – from Heathrow to UCL for example – or plan what sights to visit before arrival. In order to maintain the illusion of actually being there, the model needs to be sufficiently realistic. Improving the appearance of the buildings is not enough, we also need to simulate the human movement around them.

Including the simulation of human crowds in the static model becomes more expensive not only for the additional rendering cost, but also for the cost of the collision detection test required for the crowd.

As an example, in the busy streets of London during the day, there can easily be thousands of people moving around. Trying to perform exact collision detection using standard methods for every moving entity is very expensive and in this case is probably not necessary. If one want to see all the moving entities at the same time, he would need to be at a fair distance away, probably looking down at the model from above. In this case computing accurate collision is not necessary.

[1] This work was carried out while visiting UCL.

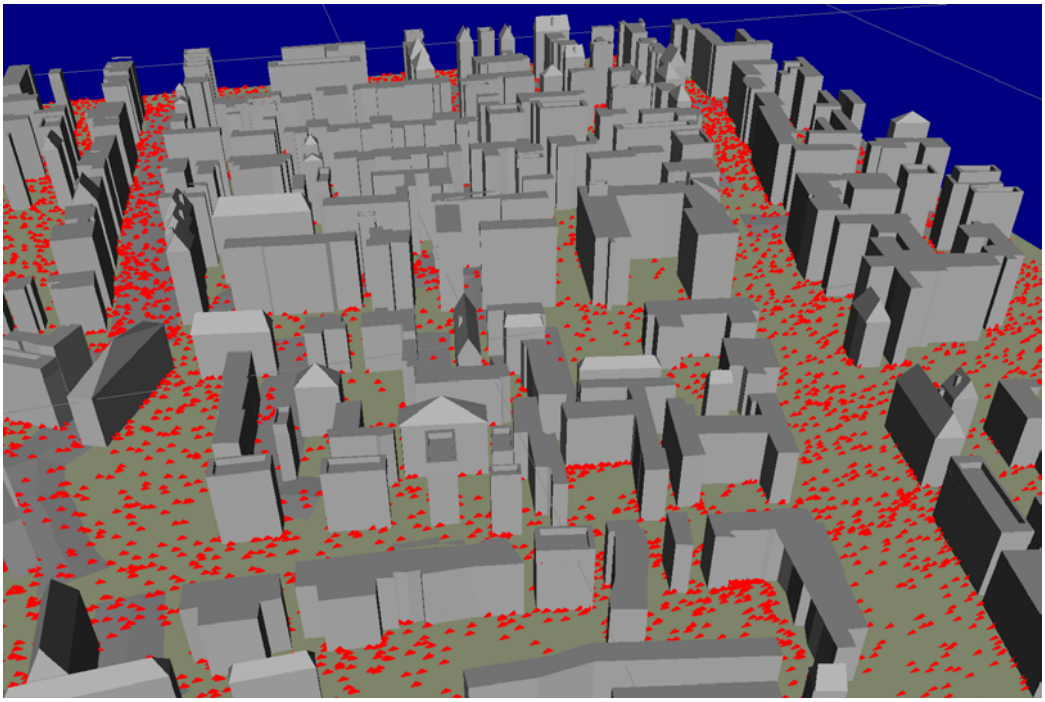

Figure 1. An example of a city model populated with 10,000 particles

In this paper we present an approximate method based on space discritisation, with the moving humans represented as single particles. In its current implementation our algo only detects interference with the static parts of the environment (i.e. no inter-particle collision) but the method is scalable and fast enough to run in real-time with tenths of thousands of particles.

In Section 2 we review some of the previous work in collision detection. In Section 3 we present an overview of the method with the implementation details following in Section 4. Experimental results are presented in Section 5 followed by a discussion and some ideas for further work.

2 Previous work

There are many techniques to detect interference between geometric objects [Lin98]. Many of them use hierarchical data structures, for example, hierarchical bounding boxes [Cohe95, Gott96], spheres trees [Hubb93], BSP trees [Nayl90] and Octrees [Same90]. However the majority of them tries to solve the harder problem of interference between complex objects. They tend to be much more precise and involved than what we need for our application. Due to the large amount of moving objects and the inherent time constrains of the application, we need to look at other approaches which can trade off small errors in exchange of greater speed and scalability.

Collision detection through discritisation of space has been used before. The most relevant work to our method is that of Myskowski [Mysk95] and Rossignac [Ross92]. Like in our approach they use graphics hardware to perform the rasterization necessary in order to find the interferences in their models, but they focus on performing this task on a small number of very complex 3D CAD objects.

When it comes to urban environments, even though the geometry is still in 3D, the movement of humans is usually restricted to follow a 2D surface, or possibly more than one if we consider elements such as bridges. Bearing in mind this and the fact that the environment itself is static, simpler solutions can be developed. Steed [Stee97] used a planar graph based on the Winged Edge Data structures for navigation in virtual environments. In Robotics, the problem was studied extensively for navigating mobile robots. Lengyel [Leng90], for example, used raster hardware to generate the cells of the configuration space used to find an obstacle-free path. Bandi and Thalmann [Band98a] also employed discritisation of space using hardware to allow human navigation in virtual environments. However they chose a different approach from ours. They use the information for automatically computing a motion path for a human in an environment with obstacles. They used a coarse subdivision on the horizontal plane and repeated that on several discreet heights, while in our case we want to consider the height of the obstacles in a more continuous way.

3 Overview

Given a city model and a set of humans represented as particles we want to detect any possible collisions between the particles and the surrounding environment. Moreover we want to be able not only to detect an encounter but also to decide its nature and act accordingly.

The overall idea of the algorithm is to create a discreet representation of the static part of the model (the *height map*) and use it to detect collisions of the moving particles with the environment.

This map stores the height at each point in the environment and it is maintained in memory. For every frame of the simulation, before moving a particle to its new position we check its current elevation against that stored in the height-map for the target-position. If these values are too different, it means that the step necessary to climb either up or down to get to the new position is too big and cannot be taken, otherwise we allow the particle to move and update its height according to the value stored in the height-map.

4 Implementation

Since our objective is the visualisation of a densely populated urban environment, we use various polygonal models representing city blocks and populate them with large numbers of particles. For each particle, we randomly define an initial obstacle-free position and a movement direction. At run-time, the particle positions are updated considering their movement direction and the presence of obstacles on their way.

The algorithm is organised in two phases: the generation of the height map and the run-time collision test.

4.1 Generating the height map

The height map is generated using standard OpenGL functions. This is done at the start of the simulation by positioning the camera over the center of the model looking down at it with the view frustum adjusted to match the model boundaries. The model is then rendered using an orthogonal projection and the resulting contents of the z-buffer, that represent a discreet map of the heights of the model, are copied into the main memory where they can be accessed in a faster way. Using OpenGL to generate the height-map allows our algorithm to be simple and very fast because of the use of dedicated hardware.

During the generation of the height-map the complexity and the scale of the model must be considered; in order to permit collision detection tests with the right order of precision, the resolution of the map has to be sufficiently high. However, it is important to notice here that although higher resolution maps are more expensive in terms of memory requirements, the speed of the height map test for each particle doesn't seem to be noticeably affected by the size of the map (see Section 5).

4.2 Collision detection and avoidance

So far we have used two different approaches for detecting and avoiding collision of the moving particles.

The principle is the same in both cases: as each particle moves in the assigned direction we check the presence of obstacles in front of it using the information stored in the height-map.

In the first case we check the position that the particle is going to occupy after the current movement; this position is computed and mapped onto the height map. If the height at this point is found to be close enough to the current height of the particle the movement is considered valid and the particle is allowed to move there. If the difference in heights is too large then a new itinerary needs to be found. This is done by gradually rotating the particle's direction in small angle steps until an obstacle-free direction is found.

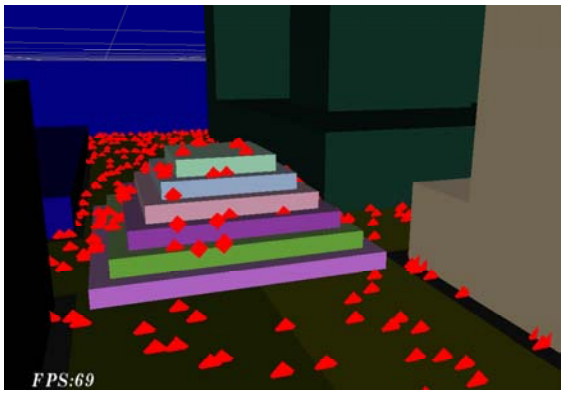

Figure 2. Particles avoid walls but climb over the smaller steps

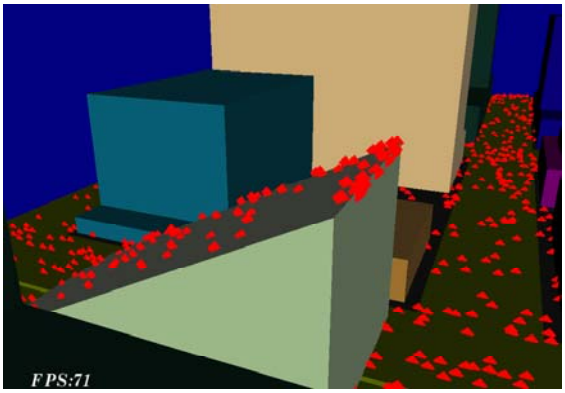

Figure 3. Particle behaviour in presence of gradual slopes

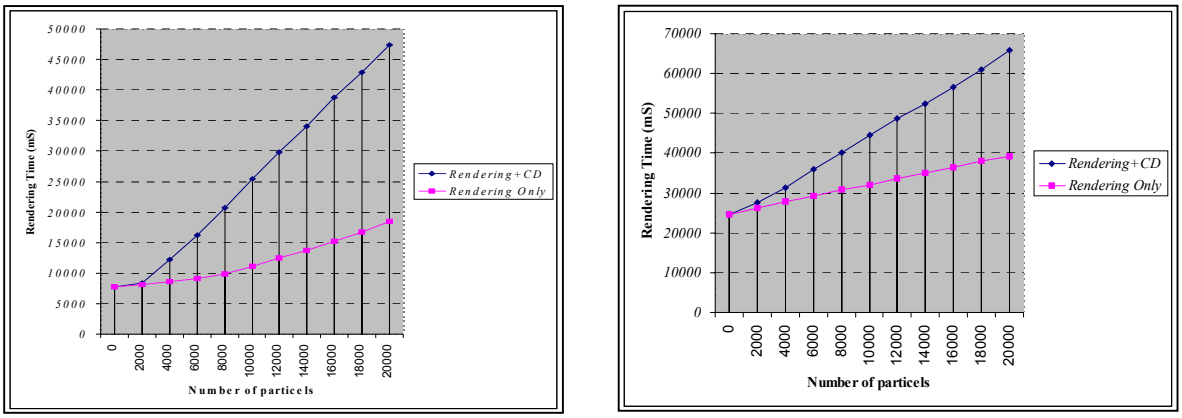

Figure 4. Time to render against increasing number of particles using two models of different complexity; on the left 2,368 polygons, on the right 32,128 polygons;

In the second case we shift the collision detection task ahead of the current particle position. Instead of checking whether our next step is possible from the current position, we check whether the i^{th} step is possible from the predicted $(i-1)^{th}$ position. If not the direction is again rotated by a small angle as in the previous case, but the position of the particle is updated anyway so that the particle starts changing direction gradually before colliding against an obstacle, producing as a result a smoother animation. On the other hand, we now need two access to the height map, making this method slower than the previous one.

The aim of this simple trial and error strategy to find a free path is to avoid querying directly the geometrical database for valid directions, in order to keep the cost of the collision test low.

In Figure 2, we can see an example of the particles behaviour. Using the height map the particles correctly detect the different dimension of obstacles, climbing on them if the steps are small enough and updating their elevation without accessing the geometrical database of the model.

Figure 3, shows the result of the algorithm in presence of gradual slopes and steep drops. The particles correctly climb the slope with smooth updates to their vertical position while still avoiding to fall over the edge.

5 Results

The system has been implemented on an Intel PC PentiumII 350 Mhz processor and an OpenGL compatible video card GeForce256, produced by NVIDIA.

We evaluated the scalability of the method in terms of increasing number of particles and geometric complexity of the urban model by measuring the following values:

1) the necessary time to render 1,000 frames with and without the collision detection procedure, varying the number of particles. Note that even when the collision detection algorithm is disabled we still draw the polygons representing the particles;
2) the necessary time to render 1,000 frames, keeping the number of particles constant (10,000) but varying the polygonal complexity of the model (and varying consequently also the resolution of the height-map).

The diagrams in Figure 4 show the results of the first tests. The X-axis reports the number of particles used while the Y-axis shows the overall time in milliseconds for the 1,000 iterations. We repeated the test using two distinct city models, the first one composed of 2,368 polygons and the second composed of 32,128 polygons. Assuming that the minimum theoretical rendering cost for a moving human is equivalent to a polygon, we have included it when computing the total rendering time.

Two important observations can be made from these graphs: the computation time necessary to perform the collision detection increases linearly with the number of humans and when models with a higher number of polygons are used the relevance of the collision detection computation becomes less significant with respect to the graphical rendering cost.

In the second test we used polygonal models of city zones with various degrees of complexity, starting from a very simple one composed of just 267 polygons (public domain model: www.microsoft.com/vrml) to a fraction of the London model[1] available at UCL composed by 32,128 polygons. The resolution of the height-map was also varied according to the complexity of the model. We started with a resolution of 256x256 for

[1] The London model was in part created using the Cities revealed data set licensed from the GeoInformation Group.

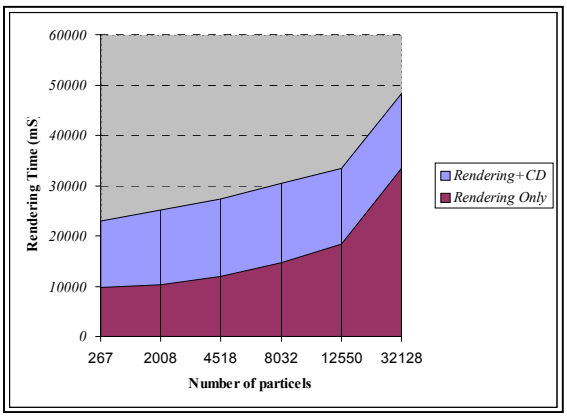

Figure 5. Time to render against increasing model complexity

the first model and gradually increased it up to 2048x1536 for the London model.

The results are shown in Figure 5. The area between the two curves corresponds to the time taken by the collision test and path finding. As we

can see it is almost constant which indicates that the collision check for the particle does not depend on the complexity of the geometrical model or the map resolution.

During the tests we obtained the height-map using a 24-bit zbuffer. In this way we can deal with 2^24 different elevation values.

Even when simulating 20,000 particles, the algorithm proved to be fast enough to reach visualisation speeds of 15 fps for the complex model (32,128 + 20,000 polygons) and 22 fps for the simpler one (2,368 + 20,000 polygons).

6 Discussion and Future Work

With the presented algorithm we are able to perform a fast collision detection in visualising densely populated models. However the current version has several limitations that we will try to address in the future. For instance, the size of the city model may be limited by the maximum resolution of the height-map. Our current OpenGL implementation does not allow a z-buffer larger than 2048x1536 pixels which may prove to be insufficient when dealing with very extensive models.

A simple way to resolve this problem is to create the height-maps of sections of the model separately and then join them together. Because maps of higher resolution imply high cost in terms of memory (in our current implementation 4 bytes are used for each pixel of the image), we could implement strategies to gain advantage from the fact that vast regions of the height-map are inaccessible (e.g. top of buildings or courtyards) and thus never used for the collision test. To exploit this we can subdivide the height-map in clusters which can be stored on the hard-disk, virtualising the RAM used to store the table. Another possibility is to implement image compression algorithms on the height-map. Then, the clusters can be stored in a much more efficient way, decompressing them the first time we need to bring back the data to RAM.

As presented here, the problem of inter-human collision is not addressed by our algorithm. Testing the inter-proximity of thousands of independent particles may represent a very expensive process using traditional approaches. We are investigating the possibility to make use of another regular map (not necessarily with the same resolution as the height-map) as a way to test for particles collision (we call it *collision-map*). Every time that the position of a particle is updated, its corresponding position in the collision-map is updated as well. By checking the content of the collision-map while planning the movement of the particles we can develop strategies to prevent them from colliding against each other. This approach, which we have implemented in a simple form, has given us so far some good results.

To render the virtual humans, we plan to use Image Based rendering techniques so as to be able to render complex human figures using a single textured polygon.

Finally, for dealing with models that have more levels of overlapping geometry we plan to extend the method to use several height-maps.

7 Conclusion

In this paper we have described a method for fast collision detection in complex city models populated with large number of moving humans. We used the graphics hardware to produce a rasterization of space which can be queried in minimal time. As a result we have shown that we can achieve collision tests for a population of thousands of individual in real time.

The algorithm presented proved to be easy to implement and adaptable to various models with different complexity.

We envisage our method to be used as part of a level-of-detail simulation. At the lowest level, where one has a full view of the city from a certain height, the interference between the moving objects might not be noticeable. As the viewer comes closer and the visible objects are reduced in number, then an extra test can be employed to decide for the inter-human collision while our method can still be used to test for collision against the static model.

References

[Band98a] Srikanth Bandi and Daniel Thalmann, Space Discretization for Efficient Human Navigation, *Proc. Eurographics '98, Computer Graphics Forum*, Vol. 17, No3, 1998, pp.195-206.

[Band98b] Srikanth Bandi and Daniel Thalmann, The Use of Space Discretization for Autonomous Virtual Humans, *Proceedings of the 2nd International Conference on Autonomous Agents* (AGENTS-98), pp. 336-337, ACM Press, May 9-13 1998.

[COVEN] Collaborative Virtual Environments Project, http://coven.lancs.ac.uk/

[Cohe95] J. Cohen, M. Lin, D. Manocha, and M. Ponamgi. I-collide: An interactive and exact collision detection system for large-scale environments, *Proceedings of ACM Interactive 3D Graphics Conference*, pages 189-196, 1995.

[Gott96] Stefan Gottschalk and Ming Lin and Dinesh Manocha, OBB-Tree: A Hierarchical Structure for Rapid Interference Detection, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pp. 171-180, Addison Wesley, August 1996.

[Hubb93] P. M. Hubbard, Interactive collision detection *Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality,* October 1993.

[LengG90] Jed Lengyel and Mark Reichert and Bruce R. Donald and Donald P. Greenberg, Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware, *Computer Graphics*, 24(4), pp. 327-335, August 1990.

[Lin98] M. Lin and S. Gottschalk, Collision Detection between Geometric Models: A Survey, Appeared in the Proceedings of *IMA Conference on Mathematics of Surfaces,* 1998.

[Mysz95] Karol Myszkowski and Oleg G. Okunev and Tosiyasu L. Kunii, Fast collision detection between complex solids using rasterizing graphics hardware, *The Visual Computer*, 11(9), pp. 497-512, Springer-Verlag, 1995

[Nayl90] B. Naylor and J. Amanatides and W. Thibault, Merging BSP Tress Yields Polyhedral Set Operations, *ACM Computer Graphics*, 24(4), pp. 115-124, August 1990.

[Ross92] Jarek Rossignac and Abe Megahed and Bengt-Olaf Schneider, Interactive inspection of solids: Cross-sections and interferences, *Computer Graphics*, 26(2), pp. 353-360, July 1992.

[Same90] H. Samet, The Design and Analysis of Spatial Data Structures, *Series in Computer Science*, Addison-Wesley, April 1990.

[Stee97] A. Steed, Efficient Navigation Around Complex Virtual Environments, *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST-97),* pp. 173-180, ACM Press, September 15-17 1997.

[Stee99] A. Steed and E. Frecon, Building and Supporting a Large-Scale Collaborative Virtual Environment, *Proc UKVRSIG99,* Salford, UK, 1999.

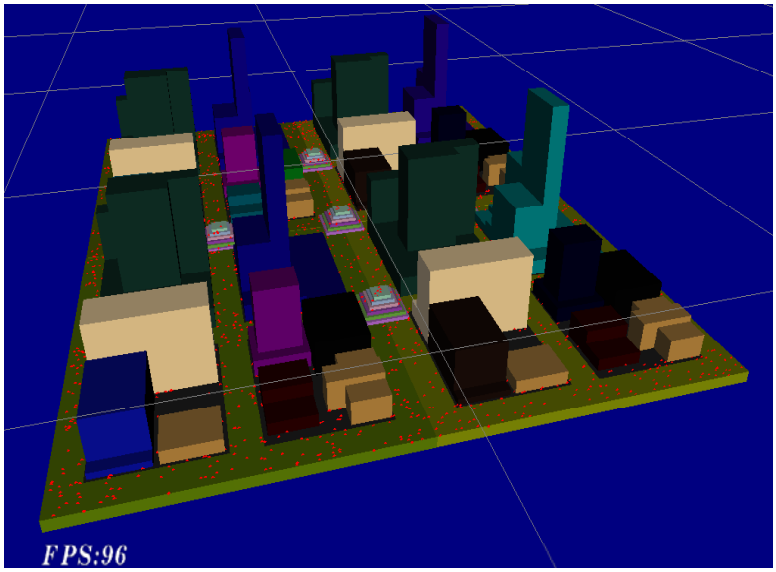

Figure 6. One of the models used with 1,000 particles

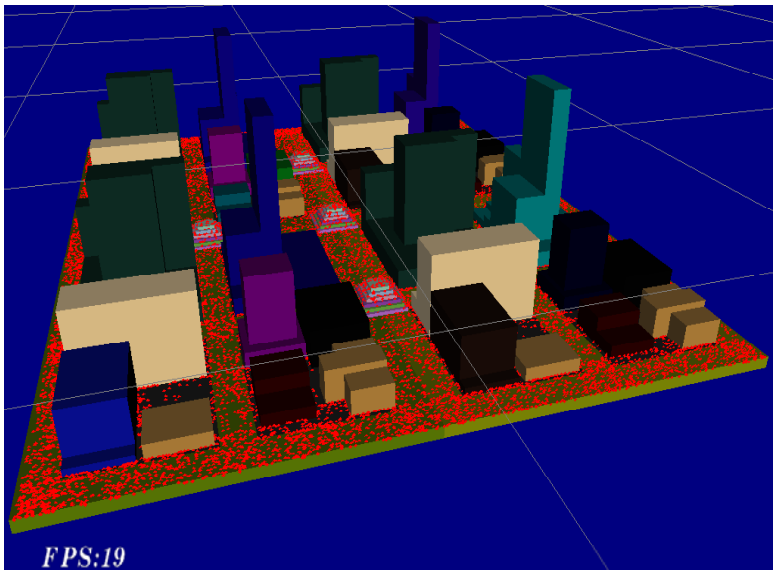

Figure 7. The same model with 20,000 particles

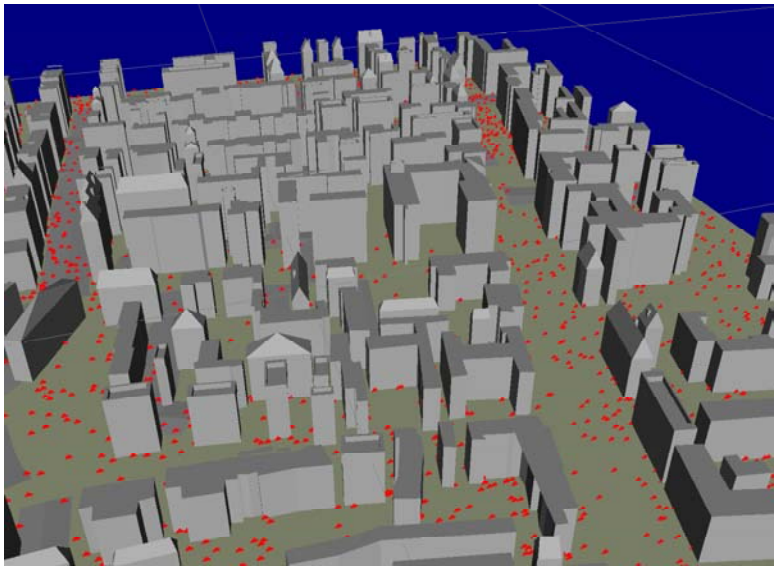

Figure 8. Part of the London model with 5,000 particles