An Evaluation of a Sketch-Based Model-by-Example Approach for Crowd Modelling

Li Guan **Burkhard C. Wünsche**

Graphics Group, Department of Computer Science
University of Auckland, New Zealand,
Email: lgua012@aucklanduni.ac.nz, burkhard@cs.auckland.ac.nz

Abstract

An increasing number of computer applications require complex 3D environments. Examples are entertainment (games and movies), advertisement, social media technologies such as "Second Life", education, urban planning, landscape design, search and rescue simulations, visual impact studies and military simulations. Many virtual environments contain thousands of similar objects such as characters, trees, and buildings. Placing these objects by hand is cumbersome, whereas an automatic placement does not allow sufficient control over the desired distribution characteristics. In previous work we presented a prototype for a sketch-based model-by-example approach to generate large distributions of objects from sketched example distributions. In this paper we present an improved algorithm and we perform a formal user study demonstrating that the approach is indeed intuitive, effective, and that it works for a large number of regular, irregular and clustered distribution patterns. Remaining limitations related to Gestalt and semantic concepts are illustrated and discussed.

Keywords: sketch-based modeling, sketch-based interface, crowd modeling, texture synthesis, mass-spring system

1 Introduction

The use of virtual environments (VE) is expanding rapidly and applications range from entertainment (games and movies), to education, social media (e.g., "Second Life"), architecture, engineering, and urban design and planning. Creating virtual environments can be a time-consuming process, especially when modelling scenes containing thousands of similar objects such as characters, trees, and buildings. Such aggregations of objects are, however, necessary to make computer generated scenes look natural and visual attractive. Placing objects individually is cumbersome, whereas using statistical models, such as regular or random patterns, does not give the user sufficient control and often looks artificial.

In previous work we showed that a model-by-example technique using sketch input is a promising approach to rapidly generate crowds (Guan & Wünsche 2011). Sketching provides complete freedom over the input, encourages creativity (Gross & Do 1996), and facilitates problem solving (Wong 1992). Large point distributions are defined by sketching the outline of the domain of the distribution (e.g., boundary of a forest) and then sketching a small example distribution, which is replicated over the domain using a combination of texture synthesis and proprietary techniques.

The technique allows the generation of a large number of different outputs, but suffers from several shortcomings, especially regarding the synthesis of clustered distributions. Also no user study was performed to confirm that the method is indeed effective in practice. In this paper we present a novel physical-based technique for optimising clustered distributions, while maintaining the characteristics of the original input. A user study confirms that the technique is intuitive, effective, efficient and fun.

In the following discussion we use the term "crowd" to mean any aggregation of objects, e.g., a forest (trees), herd (animals), village (residential buildings), or city (skyrises).

Section 2 reviews the literature in this field. Section 3 presents design requirements and our previously published algorithm for crowd modelling. Section 4 presents a new algorithm for synthesising clustered distributions and section 5 gives implementation details. The algorithm is evaluated in section 6 using experimental results and a user study. We conclude the paper with section 8, which also gives an overview of future work.

2 Literature Review

A large number of mathematical methods exists for creating point distributions. Most of them are concerned with creating random distributions with certain statistical properties. For example, Poisson Disk sampling patterns are popular in the graphics community, e.g., for rendering and illumination (Jones 2006), because they have minimal low frequencies and no spikes in the power spectrum. Quasi-Monte Carlo methods are popular in problems involving integration, such as global illumination (Szirmay-Kalos 2008) and area computation of point-sampled surfaces (Liu et al. 2006). Near uniform distributions with user defined characterisatic, such as alignment with a vector field, can be created using diffusion-advection equations (Botchen et al. 2005).

The literature offers much less references regarding point distributions for object placement. Many applications define the positions of large groups of objects using application specific physically or statistically motivated techniques, similar to the ones explained above. The popular landscape synthesis tool "Terragen" uses environmental parameters and directional controls to modify a fractal noise texture specifying the location of vegetation (Planetside Software, 2006). Procedural methods have been used for city simulations (Greuter et al. 2003). Diffusion-advection equations are useful for time-dependent processes with distance constraints such as traffic patterns (Garcia 2000). Bayesian decision processes (Metoyer & Hodgins 2004) and the partial differential equations have been used to describe local and global behaviour patterns of crowds (Treuille et al. 2006). Crowd behaviour, based on a given initial position, can be simulated using an agent-based method (Reynolds 1987,

Funge et al. 1999, Sung et al. 2005, Massive Software 2009).

Professional crowd simulation tools usually offer interfaces for randomly generating crowds over a user defined domain by specifying the size and/or density of characters (WorldOfPolygons.com 2006). A spray interface for distributing grass, trees and other objects over a terrain has been presented by van der Linden (2001).

3 Crowd Modelling Prototype

In previous research we designed a prototype for sketch-based crowd modelling (Guan & Wünsche 2011).

3.1 Requirement Analysis

The solution was motivated by an analysis of crowds in photographs and by a user study. The evaluation showed that most crowds can be characterised by the shape of their domain and their distribution pattern. Most patterns could be divided into three classes: random, regular and clustered. Within each class there is an infinite number of different distribution patterns, e.g., a regular distribution can be a rectangular grid, or a more complex repetitive arrangement. In either case the pattern can be completely regular or have different degrees of jitter in it.

Our analysis showed that a feasible way to define a large variety of crowds and collections of objects is to define its domain and an example distribution. The program must be able to differentiate between different types of distributions, such as regular, irregular and clustered, and must be able to replicate the characteristics of any such pattern without merely repeating it.

3.2 Design

Our original solution (Guan & Wünsche 2011) allows users to sketch a domain and a sample distribution using dots or short strokes. The sketched example distribution is analysed using a k-means++ algorithm (Shindler 2008) and Euclidean shortest spanning tree to determine the number of clusters in the user's input. If the user input contains only one cluster then we determine whether it is regular or stochastic by analysing the distribution of edges of a Euclidean shortest spanning tree.

For a regular input distribution we choose the smallest enclosing square and use the thus created texture image as exemplar for a Wang tiling texture synthesis algorithm (Cohen et al. 2003). We found that this algorithm preserves structures in the input texture well. For irregular distributions we choose the exemplar texture analogously, but then apply a Chaos Mosaic algorithm (Guo et al. 2000).

For a clustered input we compute the mean and standard deviation of the size of all clusters and of the distances of the points in them to the respective centers. We then generate new clusters based on these probability distributions. The clusters are then randomly placed subject to a minimum distance criterion.

The synthesis of clusters does not preserve the characteristics of the user input. For example, we are unable to replicate uniformly spaced clusters. If the user input contains randomly distributed clusters, then the synthesised result might still not look realistic, since the power spectrum of the synthesised cluster positions can vary dramatically from the exemplar.

4 Cluster Synthesis

The above described algorithm suffers from a poor synthesis of clustered input. In order to find a solution we observe that we can replace clusters with their centroids. The resulting point distribution can be used as input for the original algorithm, and the synthesised point distribution represents the location of all synthesised clusters. The generation of individual clusters is then performed as described in (Guan & Wünsche 2011). A flow chart of the resulting improved algorithm is shown in figure 1.

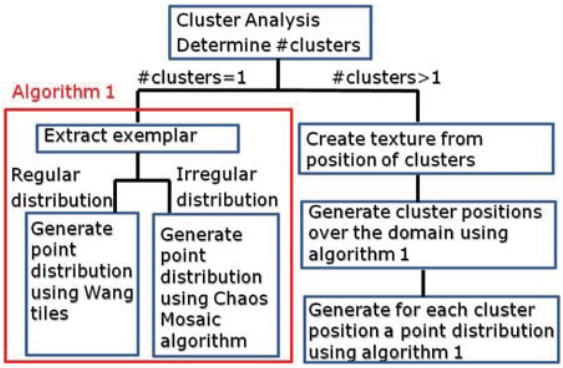

Figure 1: Flowchart of the improved algorithm for sketch-based crowd modelling.

Note that in theory an example input can contain clusters of clusters, e.g., the soldiers in an army can be arranged in, say, $n \times m$ large groups, where each large group contains $k \times l$ small groups, and each small group has soldiers standing in a grid like patterns. Such cases could be resolved by recursively applying the above algorithm, but since this case neither occurred in our user studies nor in our evaluation of image data bases, we did not implement this generalisation.

The above method does not put any constraints on the distance between clusters. As a result it is possible that cluster positions are synthesised such that clusters overlap and appear as one big cluster. This situation must be avoided, since the resulting distribution does not reflect the properties of the example input and hence looks unintuitive. The situation is illustrated in figure 2.

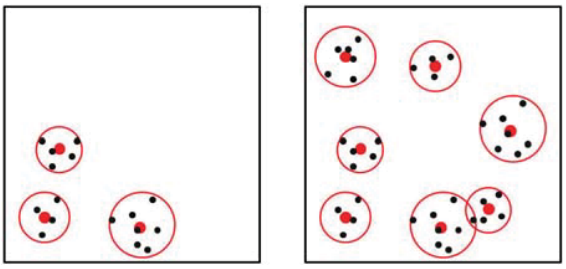

Figure 2: Left: Example point distribution (black dots) and the corresponding cluster centres (red dots) and bounding circles (red). Right: A hypothetic synthesised distribution where two cluster centres are too close resulting in overlapping clusters, which are perceived by users as one uncharacteristically large cluster.

4.1 Cluster Optimisation

In order to improve the synthesised clusters we need to find a solution which assures that cluster centres are at least a distance $d_{minClusters}$ apart. In the examples in this section we set $d_{minClusters}$ to the diameter of the largest cluster. This setting is good for illustration purposes, but does not result in a clear visual differentiation between clusters. We hence recommend to use in practice for $d_{minClusters}$ at least 1.5 times the largest cluster diameter.

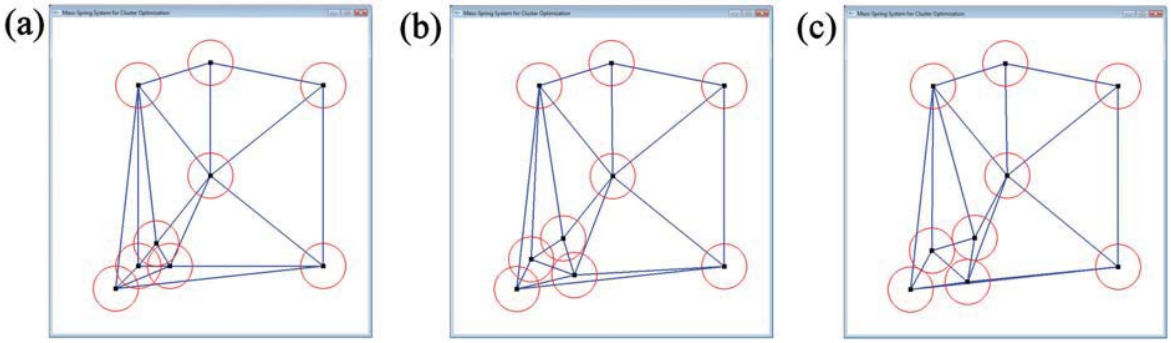

Figure 3: An irregular distribution of cluster centres (black dots) and the corresponding cluster sizes (red circles) and Delauney triangulation (blue lines). (a) The original configuration. (b) The result of applying a traditional mass-spring system where the rest length of each spring is the maximum of the original edge length and the cluster diameter. (c) The result of applying our modified mass spring system.

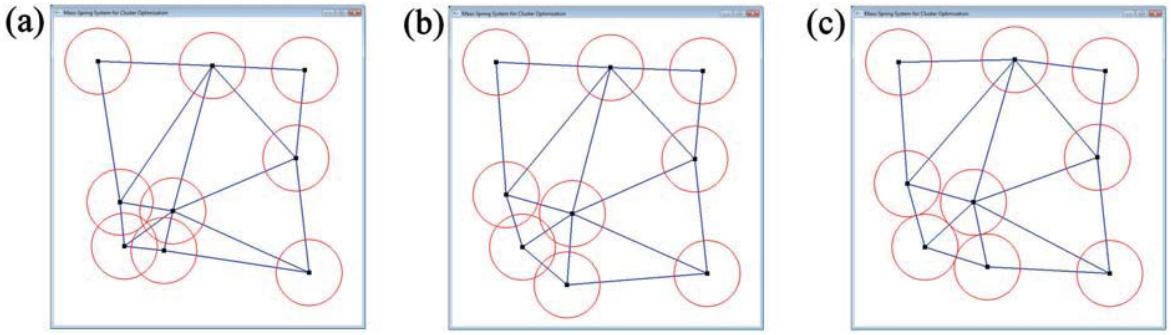

Figure 4: A jittered regular distribution of cluster centres (black dots) and the corresponding cluster sizes (red circles) and Delauney triangulation (blue lines). (a) The original configuration. (b) The result of applying a traditional mass-spring system where the rest length of each spring is the maximum of the original edge length and the cluster diameter. (c) The result of applying our modified mass spring system.

A naive approach shifting clusters until they don't overlap could significantly change the distribution pattern generated by the underlying texture synthesis method (see figure 1). We optimise cluster positions using a mass-spring system. Cluster centres represent the mass points of the mass-spring system, whereas the springs are given by the edges of the points' Delauney triangulation. Examples are given in part (a) of figure 3 and 4.

All springs are given the same spring constant $k_{standardSpring}$. If the springs' rest lengths are equal to the corresponding edge lengths of the Delauney triangulation, then no forces are generated and the system is in balance. We set a spring's rest length l_{rest} to:

$$l_{rest} = max(l_{edge}, d_{minClusters})$$

where l_{edge} is the length of the corresponding edge in the Delauney triangulation and $d_{minClusters}$ is, as explained above, the desired minimum distance between clusters.

If the distance between two cluster centres is smaller than l_{rest} then a force is generated which pushes them apart (see section 5). The result of applying this algorithm to the configuration in part (a) of figure 3 and 4 can be seen in part (b) of those figures.

Two problems can be observed:

- Clusters are still overlapping.
- Some points shift significantly from their original position, which changes the appearance of the pattern generated in the synthesis step. For example, the pattern in figure 4 (b) does not anymore look like a regular grid.

The cause of these problems is that spring forces change linearly with length changes from the rest length. It is not possible to specify that some position changes, e.g., in order to avoid cluster overlap, are more important than others.

4.1.1 Non-Linear Springs

In order to overcome the problem of overlapping clusters we use non-linear springs, where the spring force increases dramatically if the spring length is less than the desired minimum cluster distance $d_{minClusters}$. This is achieved by computing the current length l of a spring and increasing its spring constant $k_{standardSpring}$ by a factor f, if $l < d_{minClusters}$.

The physical interpretation of this modification is illustrated in figure 5: Given are two points connected by a spring with length l_{rest} and spring constant $k_{standardSpring}$. Moving the points apart results in a force pulling them together (b), whereas pushing the points closer together, results in a force pulling them apart (c). If the distance between the points is less than $d_{minClusters}$ then the spring is replaced with one of equal length, but with a much higher spring constant $f * k_{standardSpring}$. The result is a spring, which is relatively easy to extend or compress down to a length of $d_{minClusters}$, but very difficult to compress any further than this.

4.1.2 String-Springs

In order to prevent large movements away from the original cluster positions we record the position of cluster cen-

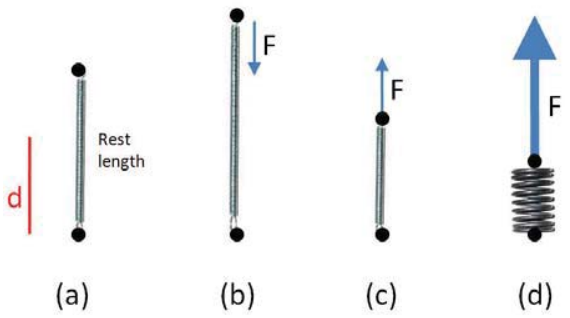

Figure 5: Physical interpretation of a non-linear spring: Two points are initially connected by a spring with rest length l_{rest} and spring constant k_{small} (a), resulting in moderate forces resisting an extension (b) or compression (c) of the spring. If the distance between points is less than d then we replace the spring with a new one, which has the same rest length, but a much higher spring constant k_{large}. The result is a spring, which is relatively easy to extend or compress down to a length of d, but very difficult to compress any further than that.

troids and connect them with springs to the current position. We want to achieve that cluster centroids can move from their original position by a distance of $d_{maxOffset}$, but any further movement should be very difficult.

We achieve this behaviour using a new concept we term *string-spring*. A string-spring can be physically imagined as a string of length $d_{maxOffset}/2$ connected to a spring with a rest length of $d_{maxOffset}/2$. An example is given in figure 6 (a). As long as the two points connected by the string-spring are less than $d_{maxOffset}$ apart no force is generated since the string can compensate for any position change of the spring, i.e., the spring is never compressed (figure 6 (b)). However, if the two points are moved further than $d_{maxOffset}$ apart, then any amount above this threshold results in an extension of the spring and a force pulling the two points together (figure 6 (c)).

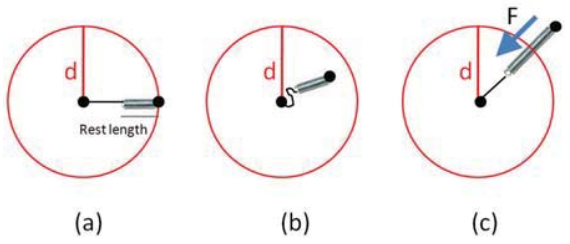

Figure 6: A string-spring can be physically imagined as a string of length $d/2$ connected to a spring with a rest length of $d/2$ (a). If the two points connected by the string-spring are less than d apart no force is generated, since the string prevents compression of the string (b). If the distance exceeds d, then any amount above this threshold results in an extension of the spring and a force pulling the two points together (c).

The results of applying our improved mass-spring system to the configuration in part (a) of figure 3 and 4 are shown in part (c) of those figures. It can be seen that in contrast to the application of the original mass-spring systems (part (b) of the figures) the clusters do not overlap and that the overall distribution pattern has a higher resemblance with the original one. This is best illustrated by figure 4, where both image (a) and (c) look like a regular grid with some jitter, whereas image (b) looks slightly random.

5 Implementation Details

We have implemented the above described algorithms using Microsoft Visual C++ and OpenGL. So far we have only integrated the generation of sketched tree objects with our crowd generation software.

5.1 Mass-Spring System

The mass-spring system is implemented as a special case of a particle system, where the n particles are the cluster centroids. Each particle P_i has a position $\mathbf{x}_i$, velocity $\mathbf{v}_i$, acceleration $\mathbf{a}_i$ and applied Force $\mathbf{F}_i$.

The applied force at any time is given by the sum of all spring forces connected to that particle. If two particles P_i and P_j with positions $\mathbf{x}_i$ and $\mathbf{x}_j$, respectively, are connected by a spring with rest length l_{ij} and spring constant k_{ij}, then the resulting force is given by Hookes Law:

$$F_{ij} = -k_{ij}(|\mathbf{x}_i - \mathbf{x}_j| - l_{ij})\frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|}$$

The second term represents the difference between the current length and the rest length of the spring, and the third term is a unit vector expressing the direction of the resulting force. In our mass-spring system the term F_{ij} is added to the total force acting on particle P_i, and the term $-F_{ij}$ is added to the total force acting on particle P_j.

The spring constant is $k_{standard}$, but increases to $f * k_{standard}$ if the distance between two particles falls below $d_{minClusters}$. For the string-springs we compute the distance between the original and current particle position. If the distance is higher than $d_{maxOffset}$ then we apply equation 5.1 with a spring constant $k_{stringSpring}$.

The constant $f * k_{standard}$ must have the highest value, since non-overlapping clusters are most important. The constant $k_{stringSpring}$ must be much larger then $k_{standard}$ in order to avoid large displacements of cluster centers. We use:

$$\begin{aligned} k_{standard} &= 1.0 \\ f &= 80.0 \\ k_{stringSpring} &= 30.0 \end{aligned}$$

5.2 Numerical Solution

The above mass-spring system results in a physical simulation where particle positions change over time. We are interested in its steady state solution, i.e., where the sum of all forces acting on particles are zero. In order to get a unique solution two requirements must be fulfilled:

- We need a fixed point of reference. This is achieved by computing the convex hull of all particle positions and fixing the positions of all points on the boundary of the convex hull. Points less then $d_{minClusters}$ apart from an already fixed point are not fixed (since we want them to move apart).
- We need to introduce a damping term to prevent the system from oscillating. The damping term will remove kinetic energy and thus enable the system to reach a steady state.

The final mass-spring system is described by the set of equations

$$F_i = m_i * \mathbf{a}_i = -k\mathbf{x}_i - c\mathbf{v}_i \qquad i = 1, \ldots, n$$

where F_i is the sum of all spring forces acting on particle P_i and c is the viscous damping coefficient, which we set to 0.2. In order to solve the system it is converted to a system of ordinary differential equations (ODEs), which can be expressed by two vectors: the current state containing the positions and velocities of all particles, and the state

derivative containing the velocities and accelerations of all particles (Witkin et al. 1994). The initial state is given by the original particle positions and by setting all velocities and accelerations to zero. The final position of all particles can then be easily computed by using an ODE solver, which terminates if the position changes in each time step are below a given threshold. Note that we use stiff springs (high spring constant) and the Euler method is hence numerically unstable. We use a fourth-order Runge-Kutta method (Press et al. 1992).

6 Results

6.1 Experimental Results

We have evaluated both the individual components of our algorithm, and the algorithm as a whole.

6.2 Exemplar Classification

Figure 7 shows the results of classifying input into regular (a), irregular (b)-(e), and clustered patterns (f)-(i). Overall the classification works well. The clustering algorithm fails if two clusters' bounding boxes overlap, e.g., nested v-shaped point distributions. This is due to the distance metric used in the k-means++ algorithm.

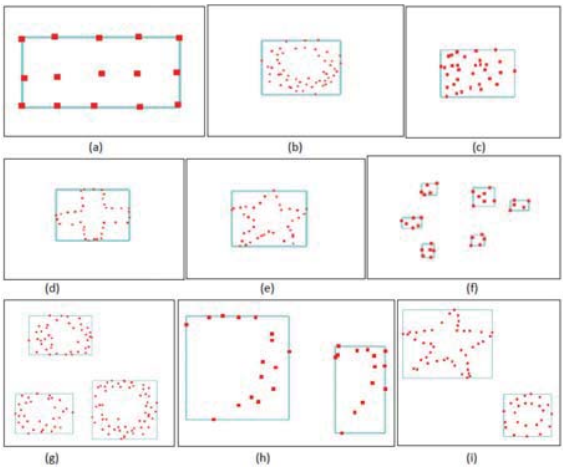

Figure 7: Examples of user input classified as regular (a), irregular (b)-(e), and clustered pattern (f)-(i).

6.3 Examples

The figures 8-10 show examples of regular, irregular and clustered input (red boxes), respectively, the resulting synthesised point distributions, and 3D scenes generated with them.

6.4 Limitations

The texture classification algorithm is unable to recognise Gestalt concepts. This was already demonstrated in figure 7, where items (d) and (e) were classified as irregular. The reason for this is, that we test for regularity by constructing an Euclidean shortest spanning tree and then analyse the distribution of its angles and edges. For a regular grid, for example, the distances to the neighbouring vertices have all approximately similar lengths. The angles with the x-axis are clustered around two values, e.g., roughly zero degree or roughly 90 degree if the grid is axis aligned. However, for the diamond shape in image (e) the distribution of edge angles is not bimodal.

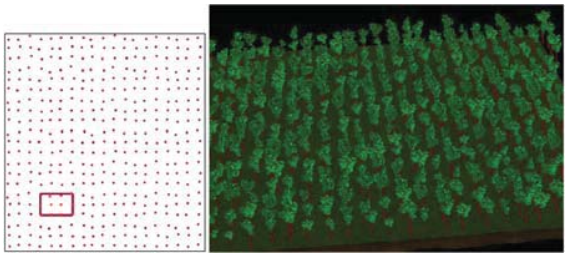

Figure 8: Example of a regular input (red box) and the synthesised point distribution (left) and model of a plantation forest generated with it (right).

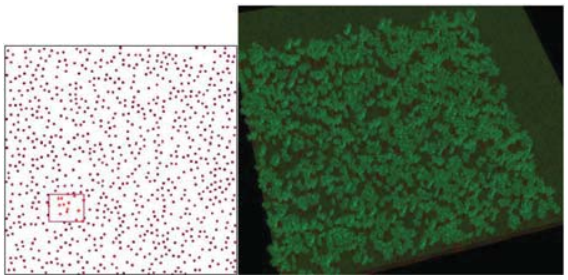

Figure 9: Example of a irregular input (red box) and the synthesised point distribution (left) and model of a natural forest generated with it (right).

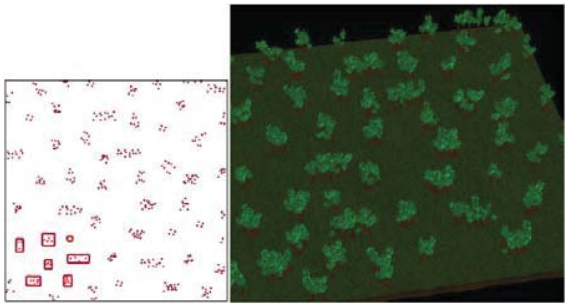

Figure 10: Example of a clustered input (red boxes) and the synthesised point distribution (left) and model of an urban park with clusters of trees generated with it (right).

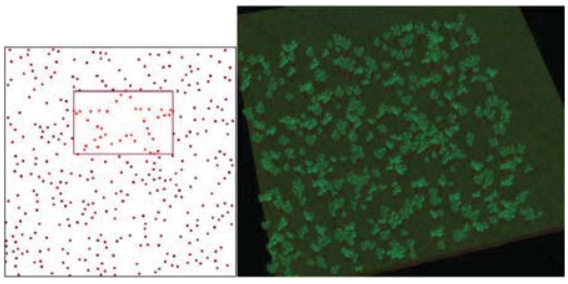

Figure 11: Example of an input point distribution (red dots in box) with Gestalt information (star shape). The input is classified as irregular and as a consequence the Chaos Mosaic algorithm is applied, which results in an unexpected output.

This problem extends to clustered distributions. For example, the input in figure 7 (g) is correctly classified as clustered. However, the point distribution within each cluster is recognised as irregular. As a result, newly generated clusters contain a random distribution of points generated with the Chaos Mosaic algorithm. An improvement over the current solution would be to recognise input distributions with Gestalt information and just repeat them using Wang tiles or another tile based algorithm. We have surveyed a wide class of texture synthesis algorithms (Guan & Wünsche 2011, Manke & Wünsche 2010), but we are not aware of any technique to replicate semantic information and Gestalt concepts in a natural manner without just repeating the input.

A second problem is that example distributions with a small number of points are insufficient to synthesise realistic looking results. As an example consider figure 12. The input consists of six clusters, which is enough for the Wang tiling algorithm to generate a realistic regular distribution of cluster centroids. For each cluster centroid a new cluster is synthesized. Since one of the input clusters is regular, the algorithm also produces some regular clusters. The number of points in the synthesised clusters depends on the variation within the input clusters. Since only one input cluster is regular all synthesised regular clusters have the same number of points, i.e., four, and they all have the same pattern (encircled in yellow).

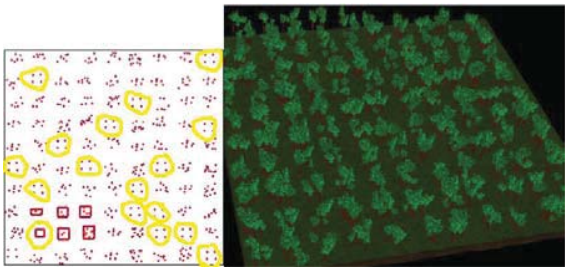

Figure 12: Example of a clustered input (red boxes) with regularly distributed cluster centroids. The clusters themselves have irregular distributions with the exception of one (encircled in yellow). The synthesised point distribution (left) suffers from repetitions, which however are barely noticeable in the resulting 3D model (right).

6.5 User Study

We evaluated the usability, efficiency and effectiveness of our algorithm with a user study. Participants had to complete three tasks:

- **Task 1:** Modelling a plantation forest with hundreds of trees (a picture of a real plantation forest demonstrating the near regular arrangement of trees was shown).
- **Task 2:** Modelling a natural forest with hundreds of trees (an aerial picture of a real natural forest demonstrating the random arrangement of trees was shown).
- **Task 3:** Modelling an urban park with clusters of trees (an aerial picture of a park with dozens of clusters of trees was shown).

We surveyed the participants after each task and at the end of the user study. Answers were recorded on a 7-level Likert scale ranging from -3 (strongly disagree) to 3 (strongly agree).

The study had 20 participants, 16 male and 4 female. All of the participants were university students or staff with six aged 16-20, eleven aged 21-25, two aged 31-35 and one between 40-45 years old. The participants were from the following departments: Computer Science (8), Commerce (5), Medicine (2), Arts (2), Education (2) and Pharmacy (1). Twelve of the participants had never used a modelling tool, four rarely and four did sometimes use one. From those users who had used 3D modelling tools the most commonly used software was Blender, Google Sketchup, and 3D Studio Max.

	Average	SD
Task 1 (Plantation forest)	38.38	19.63
Task 2 (Natural forest)	37.79	22.61
Task 3 (Park)	61.63	41.64

Table 1: Average completion times and standard deviation (SD) in seconds for the tasks 1-3.

6.6 3D Modelling Tasks

Users were told that they had to sketch the outline of the modelled scene and an example distribution indicated by dots or short sketches. Users were able to clear and restart the input if they were unsatisfied with the results. In general users required several tries to get a feeling how the resulting distribution would look like for a given input. The average completion times for the modelling tasks 1-3 are shown in table 1. It can be seen that generating regular and irregular patterns is similarly easy, but that generating clustered patterns requires at least 50% more time on average. One user in particular struggled and initially sketched the shape of each cluster. The user required help from the study supervisor and took more than 160 seconds to complete the task.

We evaluated users' experiences with the tool for three modelling tasks involving the creation of a plantation forest (regular example input), natural forest (random example input), and an urban park (clustered input). Table 2 summarises the results. It can be seen that all tasks were understood and easy-to-complete. Users strongly agreed that the tool simplified the modelling task and they were satisfied with the results. The lowest satisfaction, albeit still positive, was for modelling a clustered distribution.

A general complaint was the lack of an "eraser" tool to correct mistakes and incrementally modify the input sketch until the example input generated the desired result. Another problem was the lack of information about how the density of points in the input would be reflected in the resulting 3D scene. Several users initially drew points too close together and had to restart after they saw the resulting 3D output.

When sketching a regular input distribution several users had problems with the tool initially classifying the input as random. In a few instances users had to be told to sketch the input more carefully to make sure that it got recognised as regular input. A few users commented that the program should give feedback during the interaction.

Users struggled most when modelling clustered distributions. Several users represented clusters with circular sketches filled with points. In other cases clusters were too close together and were not recognised as individual clusters resulting in an unexpected output. Finally several users drew clusters filling most of the domain, such that no new clusters were synthesised.

6.7 General Questions

In addition to the three tasks above we allowed users to experiment and model any distribution of their choice. Examples were buildings in a city, a flower bed, fish in the ocean, rabbits in the forest, people at a festival, people in a cinema, students in a playground, and the hospital ward shown in figure 13.

We then asked questions regarding the overall usability, usefulness, and user satisfaction with the application.

	Task 1 (Plantation forest)		Task 2 (Natural forest)		Task 3 (Park)	
	Average	SD	Average	SD	Average	SD
Q1: I understood the task	2.25	0.71	2.45	0.76	2.20	1.01
Q2: The task was easy	2.50	0.69	2.50	0.61	1.70	1.66
Q3: The tool simplified the modelling task	2.65	0.59	2.70	0.57	2.25	1.45
Q4: I am satisfied with the result	2.50	0.61	2.55	0.60	2.20	1.20

Table 2: Average response on a 7-level Likert scale (from -3 to +3) and standard deviations (SD) to the questions on the left for the tasks on the top.

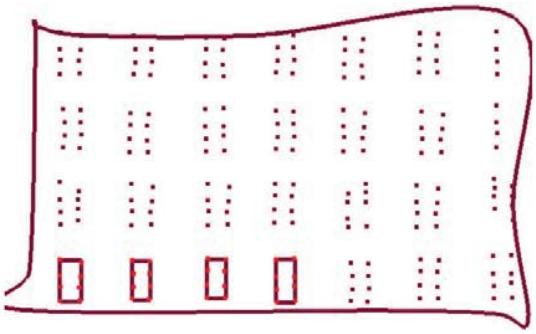

Figure 13: Example of a result of the free-drawing task in the user study: distributions of beds in a hospital ward (without partitioning walls).

The results are summarised in table 3. Overall users were satisfied with the usability of the tool. Participants were able to successfully complete modelling tasks in less than a minute, and overall users agreed that the tool was intuitive. The majority of users agreed that the application is easier to use than traditional modelling tools, but large variations in answers were observed. We believe that this might have to do with the limited functionality of the tool and the lack of feedback (error messages, help).

Despite that participants agreed that the tool is useful, and a worthwhile addition to traditional modelling tools. Most users would use the application in future, if they had the opportunity to do so. Overall users were satisfied with the application and enjoyed using it. The lowest satisfaction was with the interface. Reasons were the lack of feedback, undo and "eraser" functionalities.

7 Conclusion and Future Work

We have a presented a novel tool for modelling large distributions of objects by sketching the boundary of the occupied domain and a small sample distribution. Our work extends a previously presented prototype and includes new functionalities for synthesising clustered distributions. This is achieved by representing clusters by their centroids and using the resulting point pattern as input for a synthesis step. In order to prevent clusters from overlapping or being located too close together we developed a novel mass-spring systems. We showed that this postprocessing step was able to correct cluster distances, while still maintaining the overall characteristics of the synthesised pattern.

We demonstrated that the tool can successfully generate a large number of regular, irregular and clustered distributions. Gestalt and semantic properties of input patterns cannot be synthesised and hence result in unexpected output. If the example input has too few points this can lead to repetitive patterns.

A user study demonstrated that participants were satisfied to very satisfied with the application. Regular and irregular distributions could be generated in less than 40 seconds without help. Some users had problems with generating clustered output. The most commonly mentioned problems were related to the interface, i.e., lack of feedback, no undo functionality, no "eraser", and a lack of immediate feedback of the effect of user input on the resulting 3D scene.

In future work we want to increase the range of reproducible input distribution patterns and in particular incorporate Gestalt concepts. In addition we want to fully integrate this crowd modelling software into our "LifeSketch" software for prototyping virtual environments (Olsen et al. 2011, Yang & Wünsche 2010).

References

Botchen, R. P., Weiskopf, D. & Ertl, T. (2005), Texture-based visualization of uncertainty in flow fields, *in* 'Proceedings of IEEE Visualization', pp. 647–654.

Cohen, M. F., Shade, J., Hiller, S. & Deussen, O. (2003), 'Wang tiles for image and texture generation', *ACM Trans. Graph.* **22**(3), 287–294.

Funge, J., Tu, X. & Terzopoulos, D. (1999), Cognitive modeling: knowledge, reasoning and planning for intelligent characters, *in* 'Proc. of the 26th annual conference on Computer graphics and interactive techniques (SIGGRAPH '99)', ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, pp. 29–38.

Garcia, A. L. (2000), Physics of traffic flow, *in* 'Numerical Methods for Physics', 2nd edn, Prentice Hall, chapter 7.

Greuter, S., Parker, J., Stewart, N. & Leach, G. (2003), Real-time procedural generation of 'pseudo infinite' cities, *in* 'Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia (GRAPHITE '03)', ACM, New York, NY, USA, pp. 87–ff.

Gross, M. D. & Do, E. Y.-L. (1996), Ambiguous intentions: a paper-like interface for creative design, *in* 'Proceedings of the 9th annual ACM symposium on User interface software and technology (UIST '96)', ACM, New York, NY, USA, pp. 183–192.

Guan, L. & Wünsche, B. C. (2011), Sketch-Based Crowd Modelling, *in* 'Proceedings of the 12th Australasian User Interface Conference (AUIC 2011)', pp. 67–76. http://www.cs.auckland.ac.nz/~burkhard/Publications/AUIC2011_GuanWuensche.pdf.

Guo, B., Shum, H., & Xu, Y.-Q. (2000), Chaos mosaic: Fast and memory efficient texture synthesis, Technical report MSR-TR-2000-32, Microsoft Research. http://research.microsoft.com/pubs/69770/tr-2000-32.pdf.

Jones, T. R. (2006), 'Efficient generation of poisson-disk sampling patterns', *Image Rochester NY* **11**(2), 1–10.

Liu, Y.-S., Yong, J.-H., Zhang, H., Yan, D.-M. & Sun, J.-G. (2006), 'A quasi-monte carlo method for computing areas of point-sampled surfaces', *Comput. Aided Des.* **38**, 55–68.

	Average	SD
Q 1.1: The modelling process is intuitive	2.00	0.86
Q 1.2: I quickly learned how to use the tool	2.35	0.81
Q 1.3: I easily remembered the tool's functionalities	2.10	1.17
Q 1.4: I quickly learned how to use all functionalities	2.10	1.07
Q 1.5: The tool is easy to use	2.35	0.88
Q 1.6: The tool is easier to use than traditional modelling tools (e.g., Blender)	1.50	1.51
Q 2.1: The tool is useful	2.15	0.75
Q 2.2: The tool is more useful for generating large distributions than traditional modelling tools	2.11	0.81
Q 2.3: The tool is a useful addition to traditional modelling tools	2.16	0.90
Q 2.4: The distributions generated with the tool look realistic	2.10	0.91
Q 2.5: The distributions generated with the tool look as expected	2.15	0.99
Q 2.6: The distributions generated with the tool look as I wanted	2.30	0.66
Q 2.7: The tool made the modelling of large distributions more effective	2.55	0.69
Q 2.8: I would use the tool frequency, if I could	1.95	0.94
Q 3.1: Overall I am satisfied with the interface of the tool	1.95	1.10
Q 3.2: Overall I am satisfied with the functionalities of the tool	2.10	1.12
Q 3.3: Overall I am satisfied with the results achieved with the tool	2.20	0.95
Q 3.4: The tool was fun to use	2.55	0.60

Table 3: Questions regarding usability (Q1.1 - Q1.6), usefulness (Q2.1 - Q2.8), and user satisfaction (Q3.1 - Q3.4). The columns give the average response on a 7-level Likert scale (from -3 to +3) and their standard deviation (SD).

Manke, F. & Wünsche, B. C. (2010), Fast spatially controllable multi-dimensional exemplar-based texture synthesis and morphing, *in* M. P. J. A. H. Ranchordas, A., ed., 'Computer Vision and Computer Graphics', Vol. 68 of *Communications in Computer and Information Science*, pp. 21–34. http://www.cs.auckland.ac.nz/~burkhard/Publications/SpringerCCIS2009MankeWuensche.pdf.

Massive Software (2009), 'Homepage'. http://www.massivesoftware.com.

Metoyer, R. A. & Hodgins, J. K. (2004), 'Reactive pedestrian path following from examples', *The Visual Computer* **20**, 635–649.

Olsen, D. J., Pitman, N. D., Basakand, S. & Wünsche, B. C. (2011), Sketch-based building modelling, *in* 'Proceedings of GRAPP 2011', pp. 119–124. http://www.cs.auckland.ac.nz/~burkhard/Publications/GRAPP2011_OlsenEtAl.pdf.

Planetside Software, (2006), 'Terragen'. http://www.planetside.co.uk/terragen/tgd/tg2faq.shtml#faq34.

Press, W. H., Vetterling, W. T., Teukolsky, S. A. & Flannery, B. P. (1992), *Numerical Recipes in C - The Art of Scientific Computing*, 2nd edn, Cambridge University Press. http://www.library.cornell.edu/nr/bookcpdf.html.

Reynolds, C. W. (1987), Flocks, herds and schools: A distributed behavioral model, *in* 'SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques', ACM, New York, NY, USA, pp. 25–34.

Shindler, M. (2008), Approximation algorithms for the metric k-median problem, Technical report, UCLA, Los Angeles, CA. http://cs.ucla.edu/~shindler/shindler-kMedian-survey.pdf.

Sung, M., Kovar, L. & Gleicher, M. (2005), Fast and accurate goal-directed motion synthesis for crowds, *in* 'Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation (SCA '05)', ACM, New York, NY, USA, pp. 291–300.

Szirmay-Kalos, L. (2008), *Monte Carlo Methods in Global Illumination - Photo-realistic Rendering with Randomization*, VDM Verlag, Saarbrücken, Germany, Germany.

Treuille, A., Cooper, S. & Popović, Z. (2006), 'Continuum crowds', *ACM Trans. Graph.* **25**(3), 1160–1168.

van der Linden, J. (2001), Interactive view-dependent point cloud rendering, *in* 'Proceedings of IVCNZ 2001', pp. 1–6. http://www.cs.auckland.ac.nz/~jvan006/papers/pointcloudrendering_final.ps.gz.

Witkin, A., Kass, M. & Baraff, D. (1994), An introduction in physically based modeling, *in* 'SIGGRAPH '94, course notes #32 - An Introduction to Physically Based Modeling', ACM SIGGRAPH. Held in Orlando, Florida, 24 – 29 July.

Wong, Y. Y. (1992), Rough and ready prototypes: lessons from graphic design, *in* 'Posters and short talks of the 1992 SIGCHI conference on Human factors in computing systems (CHI '92)', ACM, New York, NY, USA, pp. 83–84.

WorldOfPolygons.com (2006), 'CrowdIT'. http://www.crowdit.worldofpolygons.com/.

Yang, R. & Wünsche, B. C. (2010), LifeSketch - A Framework for Sketch-Based Modelling and Animation of 3D Objects, *in* 'Proceedings of the Australasian User Interface Conference (AUIC 2010)', pp. 1–10. http://www.cs.auckland.ac.nz/~burkhard/Publications/AUIC2010_YangWuensche.pdf.