Journal for Geometry and Graphics
Volume 15 (2011), No. 2, 169–179.

Algorithms for Generation of Irregular Space Frame Structures

Franz Gruber, Günter Wallner

Department for Geometry, University of Applied Arts Vienna
Oskar Kokoschka Platz 2, A-1010 Wien, Austria
email: franz.gruber@uni-ak.ac.at

Abstract. Complex space frames with respect to aesthetics and stability are an important factor in contemporary architecture. Obviously there are many different ways to generate spatial structures, especially if randomness affects the generating process. In this work we present two algorithms to generate irregular space frames inside arbitrary (including non-convex) boundary volumes with predefined support areas. The resulting structures are intended as input for a genetic algorithm which optimizes the static stability. The first algorithm uses 3D-Voronoi structures as a starting point, which makes sense in terms of the framework's load capacity. The second approach uses a repulsive force field for the calculation of curve-skeletons of three-dimensional objects.

Key Words: space frames, structures, 3D Voronoi tessellations, skeletonization, vector field

MSC 2010: 00A67, 68U05

1. Introduction

The current methods of structural design are usually based on structures with a high degree of regularity (see, for example, [4] for numerous case studies). However, there is a desire for irregular complex space frames in contemporary architecture. The goal of the project *"Algorithmic Generation of Complex Space Frames"*[1] is to analyze new and innovative approaches to develop irregular and at the same time effective structures. Part of this project was the development of algorithms to generate irregular space frames inside arbitrary (including non-convex) boundary volumes with predefined support areas. At this point we should stress that the algorithms, as described in this paper, focus mainly on the framework's topology and not on its load capacity. The task of optimizing the load capacity is done in a following step by a genetic algorithm (see A. Hofmann et al. [9] for a basic description).

[1]Austrian Science Fund Grant No. L358

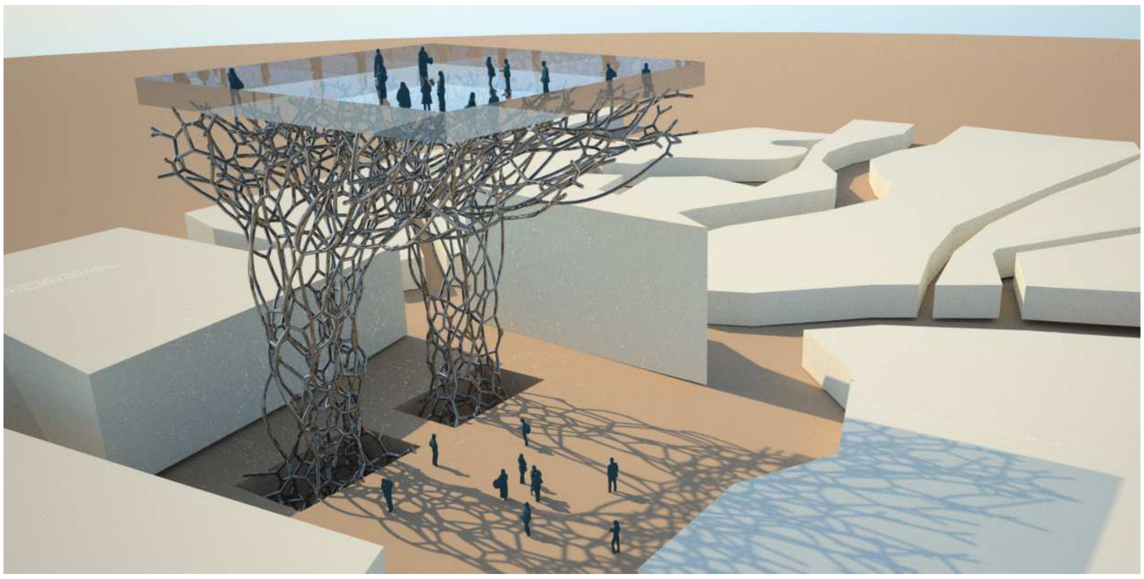

Figure 1: A viewing platform supported by a framework of Voronoi paths

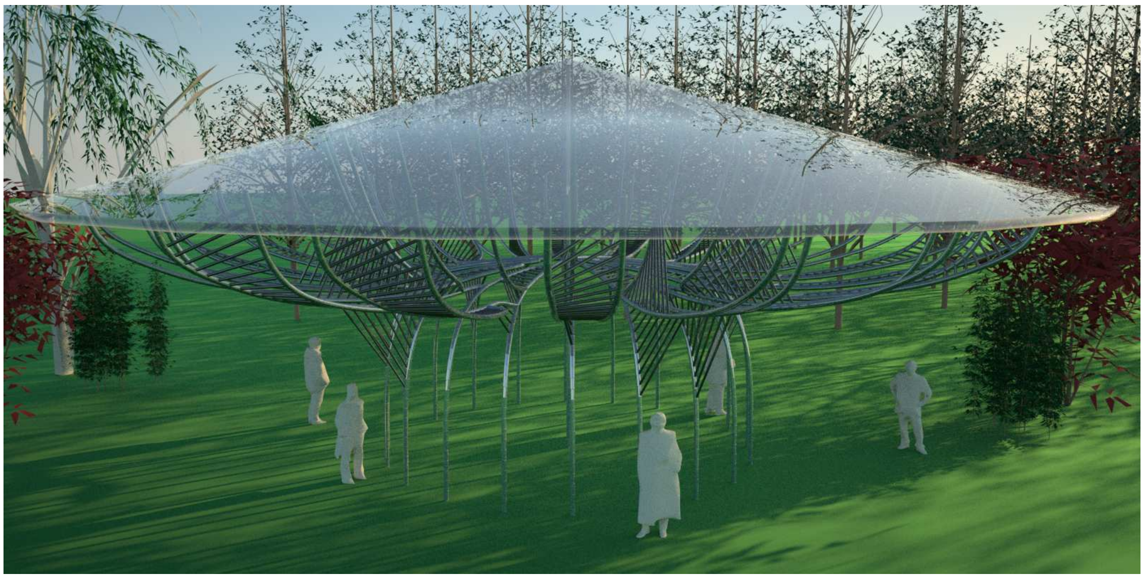

Figure 2: A pavilion were the girders where laid out with the skeletonization algorithm

Space frames were independently developed by Alexander Graham Bell, who invented space frames assembled from tetrahedral frames around 1900 and by Buckminster Fuller, who's investigations five decades later led to the creation of the famous geodesic dome. Nowadays, not only regular but also irregular space frames are becoming increasingly popular in architectural design. Notable buildings are for example the *Biosphere 2* in Oracle, Arizona, the *Eden Project* in the United Kingdom, the *Federation Square* in Melbourne and the *Beijing National Aquatics Center*.

Obviously there are many different ways to generate spatial structures, especially if randomness affects the generating process. In this work we present two algorithms which, based on different experiments, turned out to be promising.

- The first method uses 3D Voronoi structures as a starting point, which are known to produce statically rigid structures of space-filling tetrahedra [3].

- The second method uses a repulsive force field for the calculation of the structure and was influenced by the work of N.D. Cornea et al. [6].

Figures 1 and 2 show concept renderings for buildings where the framework was calculated with the two described algorithms.

This paper is structured as follows: Section 2 reviews related work in the area of irregular space frames. In Section 3 the bounding volume is discussed and Section 4 as well as Section 5 present the two algorithms. The paper is concluded in Section 6.

2. Related work

Because contemporary landmark architecture — as pointed out in [7] — continually moves away from economic considerations towards increasing numbers of building elements that are unique to the individual project, research on irregular structures has increased over the past years.

For example, A. Kanellos [11] addressed a problem similar to ours, where a certain volume has to be filled with a structural space frame network lattice consisting of a given number of nodes. The author employs a particle-spring system where the connectivity between the particles is not predetermined but established dynamically. The system uses only local rules of inter-particle interaction so that the particles are able to generate crystal-like lattices through self-organisation.

P.M. Canzarra [3] also starts with a population of points in space but uses a model derived from bone accretion, whose mechanisms are relatively well known and simple, and produce structures with good static stability. As in our case, P.M. Canzarra was not interested in finding optimal solutions but in finding challenging and creative ones. A Delauney triangulation (the dual of the Voronoi tessellation) was used as a starting structure.

P.L. Jaworski [10] published a method to “grow” a structure that supports a building by providing initial seeds and the volumes to be supported. Influenced by the concept of phototropic growth, stems originating at the initial seeds grow vertically upwards and avoid obstacles and therefore entwine existing volumes. During growth the stems are connected to nearby points to ensure static stability. The resulting structures look similar, although denser, than structures produced by our first algorithm.

T. Fischer [7] proposed a method to create apparently irregular structures from relatively small sets of identical parts, by combining a highly regular space-filling structure with a bottom-up generative procedure.

3. Boundary volume

Because the space frames have to be generated inside a given (not necessarily convex) polygonal boundary volume, and multiple solutions which can be used as input for the genetic algorithm have to be generated, an efficient representation of the boundary volume is essential. In fact, the data structure has to allow fast intersections with a ray and admit tests about whether a point is inside or outside. A kd-tree [2] is therefore used as a spatial data structure to allow fast traversal of the mesh for intersection tests. Furthermore, it has to be possible to define certain faces as supporting areas, in other words, areas where support points of the framework can be placed.

4. Voronoi paths

The algorithm starts by distributing points inside the bounding box of the given boundary volume to be used as basis for a 3D Voronoi tessellation. Afterward, the tessellation is cropped at the boundary volume. Then a given number of paths is traced along the cropped tessellation between two points from different supporting areas (*Voronoi paths*). Finally, the paths are smoothed and yield the irregular structure. The viewing platform in Fig. 1 was constructed with this algorithm. For the sake of clarity, we will explain the process in two dimensions.

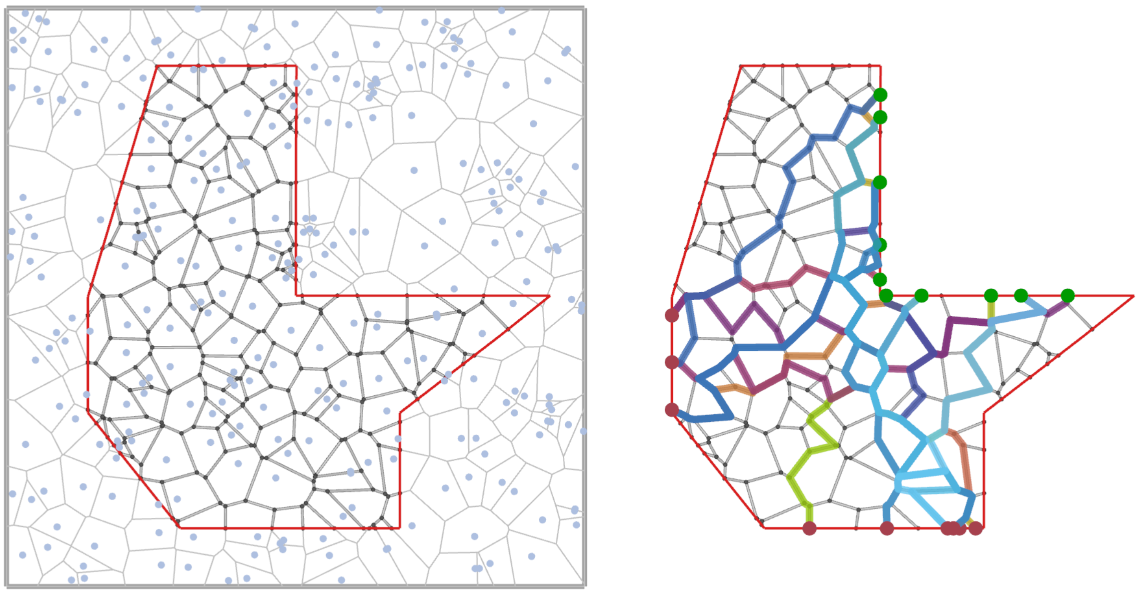

Figure 3: Left: The Voronoi tessellation is cropped at the boundary (red) which serves as the *traffic system* for the path finding. Right: Boundary points (red and green dots) from different boundary areas are connected by individual paths.

4.1. Preprocessing

In the first step, points are uniformly distributed inside the bounding box of the given boundary volume. The density is an important parameter for the fineness of the final framework. These points are used as generating points to calculate a 3D Voronoi tessellation. For this calculation the points are passed to the software package *qhull* [1] and the edges of the returned tessellation are then cropped at the boundary volume. The intersection points are called boundary points in what follows. These steps are depicted in Fig. 3 (left). A three dimensional example is shown in Fig. 4.

4.2. Voronoi path finding

Obviously, the cropped tessellation does not meet the condition that its boundary points are solely located at the predefined support areas. And from an aesthetic point of view it does not satisfy the required irregularity. However, we use this structure as a kind of *traffic system* to extract Voronoi paths between two randomly chosen support points p_A and p_B from different support areas. Finding a path between p_A and p_B is not a well-defined task. However, in the

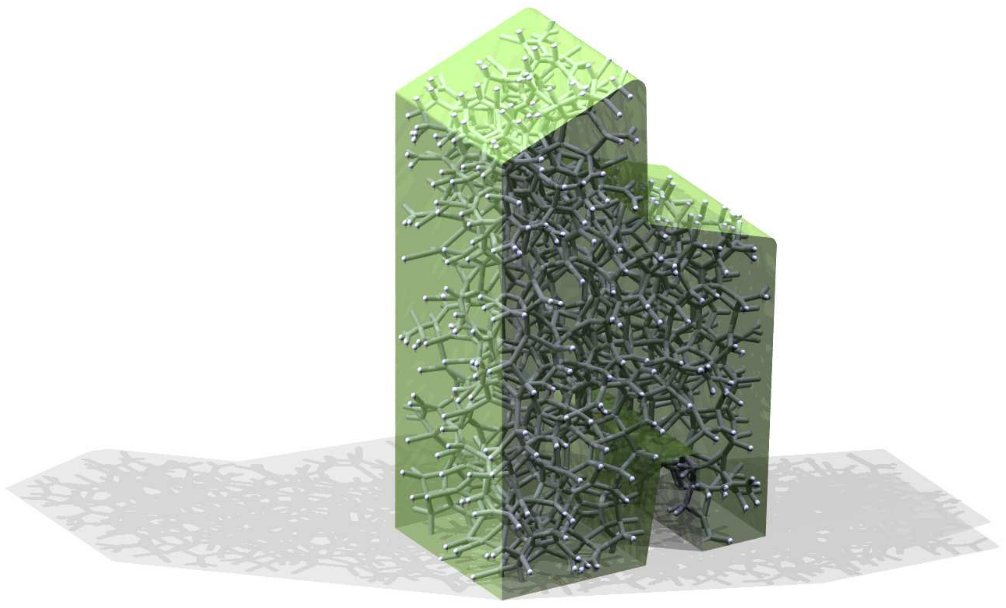

Figure 4: 3D Voronoi tessellation cropped at a non-convex boundary surface

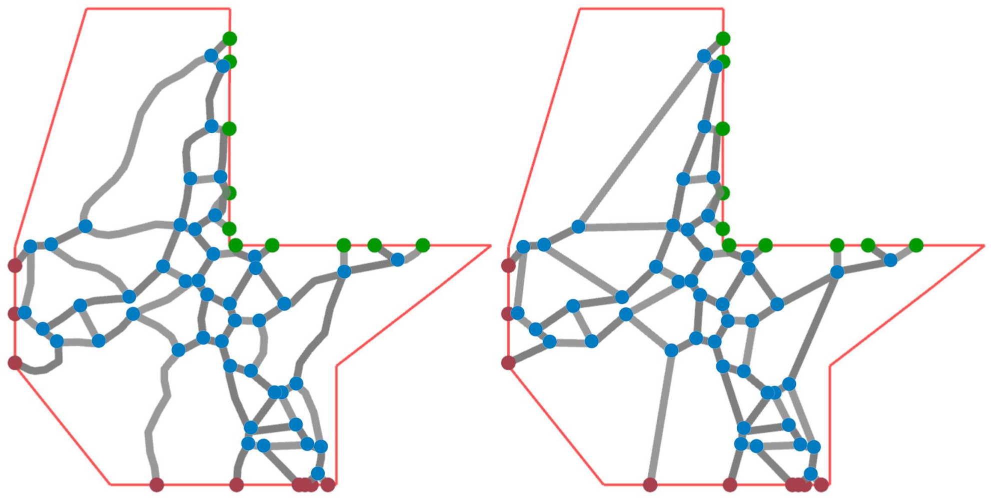

Figure 5: Left: The network from Fig. 3 after a few smoothing steps. Right: A fully straightened network. In both cases crossing points (blue) were fixed at their initial position.

current implementation we try to connect these points in a preferably short way, by using the following method.

Starting at position p_A, the path follows at each crossing point p_C that adjacent edge which has the smallest angle to the target direction $p_B - p_C$. This way we obtain a randomized network of crossing lines which partially coincide, bifurcate or converge (see Fig. 3 (right) and Fig. 6 (left) for an example in 3D). This also means that one and the same point can occur multiple times in this network. However, to obtain a graph structure which is required for the following force-directed smoothing algorithm any duplicate coincident points need to be removed.

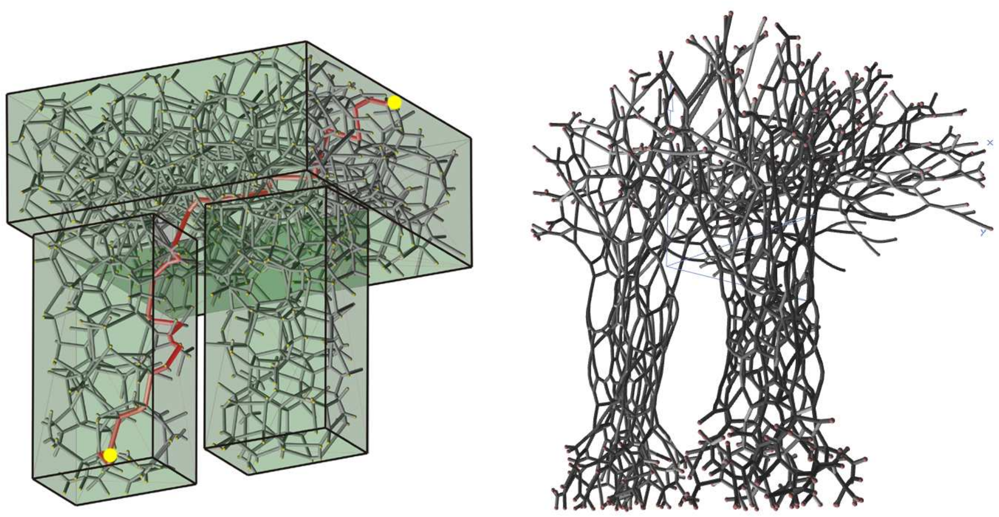

Figure 6: Left: With the help of a random 3D-Voronoi structure we define a Voronoi path (red) between two support points located at the top and bottom (yellow). Right: Complex structure composed of many smoothed Voronoi paths (see also Fig. 1).

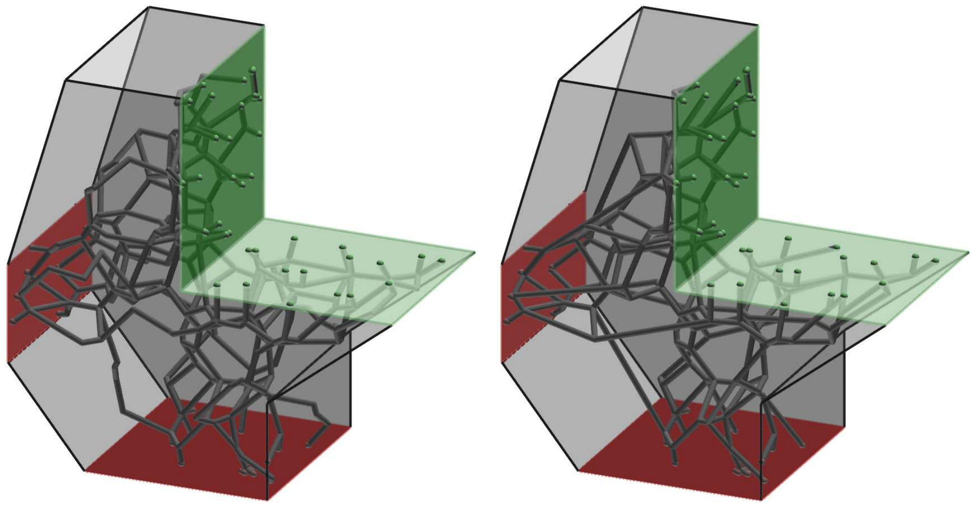

Figure 7: Left: Voronoi paths from the red to the green support area. Right: The fully straightened structure after smoothing with fixed crossing points.

4.3. Smoothing

At this point the network already has the topology of the final framework. However, the current network is characterized by zigzag lines which do not make much sense either from a statical or from an aesthetic point of view. For this reason the network is smoothed iteratively by replacing edges with elastic springs. For each vertex v a displacement $\mathbf{v}^{\mathbf{disp}}$ is stored which is set to zero at the beginning of each iteration. Then, the algorithm loops through each edge with incident vertices v_1 and v_2 and adds $\epsilon \cdot \mathbf{d_0}$ to $\mathbf{v_1^{disp}}$, respectively subtracts it from $\mathbf{v_2^{disp}}$, where $\mathbf{d_0} = (\mathbf{v_2^{pos}} - \mathbf{v_1^{pos}}) / \left\| \mathbf{v_2^{pos}} - \mathbf{v_1^{pos}} \right\|$ and ϵ is a small constant. At the end of each iteration the displacement $\mathbf{v}^{\mathbf{disp}}$ of each vertex is added to its position $\mathbf{v}^{\mathbf{pos}}$. For boundary points the last step is omitted to keep their initial positions fixed. Conventional space frame structures feature completely straight girders and can be best achieved by additionally locking the position of *crossing points* (a vertex with at least three adjacent edges).

In general, the higher the number of iterations, the more straight the network will become. Figure 5 compares a slightly smoothed with a fully straightened network. Further three dimensional examples are shown in Fig. 6 (right) and Fig. 7.

5. Skeletonization

The second approach follows the work of N.D. Cornea et al. [6], who use a repulsive force field for the calculation of curve-skeletons of three-dimensional objects. Although the connection to architecture might not seem obvious at first because their research was originally targeted to areas like virtual colonoscopy or animation, we found it appropriate for our purposes. The method uses a generalized potential field [5] to generate a discretized vector field inside an object by charging the object's boundary.

5.1. Overview

The algorithm starts by distributing point charges on the triangulated surfaces of the boundary volume. Afterward a discretized vector field is calculated within the surface's bounding box. Although this would not be strictly necessary, it accelerates the numerical integration later in the process and eases the computation of critical points. Once the vector field is established, the particle trajectories, starting from the supporting areas toward a critical point, are calculated with numerical integration. Because many of these trajectories are running practically parallel to each other they are merged into one to avoid unaesthetic clutter. Finally, cross-links depending on an angle-threshold are inserted into the existing space frame structure. Figure 8 illustrates these steps.

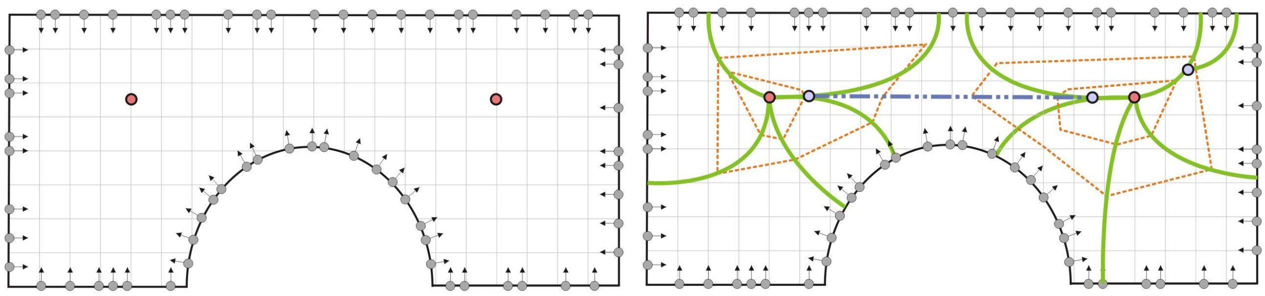

Figure 8: Left: After point charges (circles with arrows) have been uniformly distributed on the object's boundary, a discretized vector field (shown schematically as gray grid) is derived which contains at least one critical point (red circles). Right: The trajectories of particles (green) which originate on the object's boundary are the core of the final framework. Separate components are joined by straight lines (blue, dot-dashed) which connect the two closest crossing points (blue circles) to ensure that the entire structure is a coherent whole. Cross-links (orange, dashed) are inserted to ensure better stability.

5.2. Preprocessing

For each triangle of the boundary surface point charges q are placed at the vertices of the triangle and on the center between its barycenter v_{bc} and each triangle vertex v_1, v_2, v_3. The placement is repeated recursively for each sub-triangle (v_{bc}, v_1, v_2), (v_{bc}, v_2, v_3) and (v_{bc}, v_3, v_1) until a given subdivision level is reached. Further, all point charges are displaced slightly in the direction of the corresponding outward pointing surface normal. This is necessary because if the particles start at the boundary surface they should not move outside the boundary volume right away.

Based on the bounding box of the volume a discretized vector field is constructed where the number of divisions in each direction depends on the associated side length. At each cell center a vector

$$\mathbf{v} = c \cdot \sum_{i=0}^{n} \left(\frac{q_i}{\|\mathbf{d}_i\|^m} \right) \cdot \mathbf{d}_i \tag{1}$$

is calculated, where n is the number of all point charges and $\mathbf{d_i}$ is the vector pointing from the cell center to the position of point charge i. The exponent m is a quantity for how fast the influence of a point charge decreases with distance and c is a small constant.

In a discretized vector field a critical point may only occur in cells where all three components of $\mathbf{v}$ pass through 0, which in case of trilinear interpolation can be found with a simple heuristic as described in [8]. If for each component of the force vector at each cell vertex both negative and positive values exist then the components must change sign somewhere inside the cell and the cell is a potential candidate for containing a critical point. If the condition is fulfilled then the cell is recursively subdivided and the test is repeated for each sub-cell until either the test fails or a maximum number of subdivisions is reached. As noted by A. Globus et al. [8] this is only a necessary condition and the cell must not contain a critical point. They therefore use Newton's method to better estimate the location of the critical point after a fixed number of subdivisions. However, in our case the exact location of the critical point is not necessary and incorrect classifications do not interfere with the working of the algorithm. Locating the critical points is the most time consuming task in the preprocessing step. However, the preprocessing must only be performed once and does not need to be repeated to generate different space frames for a given boundary volume. Figure 8 (left) shows the system after the preprocessing is finished.

5.3. Particle trajectories

Once the pre-process has finished, paths through the vector field are traced which build the foundation of the final framework. This process starts by placing particles randomly on the supporting areas. For each particle the trajectory is calculated by explicit Euler integration [2].

Because all particle trajectories have to end at a critical point (or to be more specific at an attracting node), and there must be at least one critical point within the closed boundary volume, the integration is aborted if the last position is in proximity to such a critical point. This point is added as the last point to the particle trail. These paths are shown in green in Fig. 8 (right). Because the time step Δt for numerical integration is usually relatively small it is not practical to add each position to the final path. Therefore a minimum distance $\epsilon_d > \Delta t$ between two points must be fulfilled.

[2]More precise integration schemes, like Runge Kutta 4th order integration can also be used but the additional effort may not be necessary because paths that accurate are not required for the matter in hand.

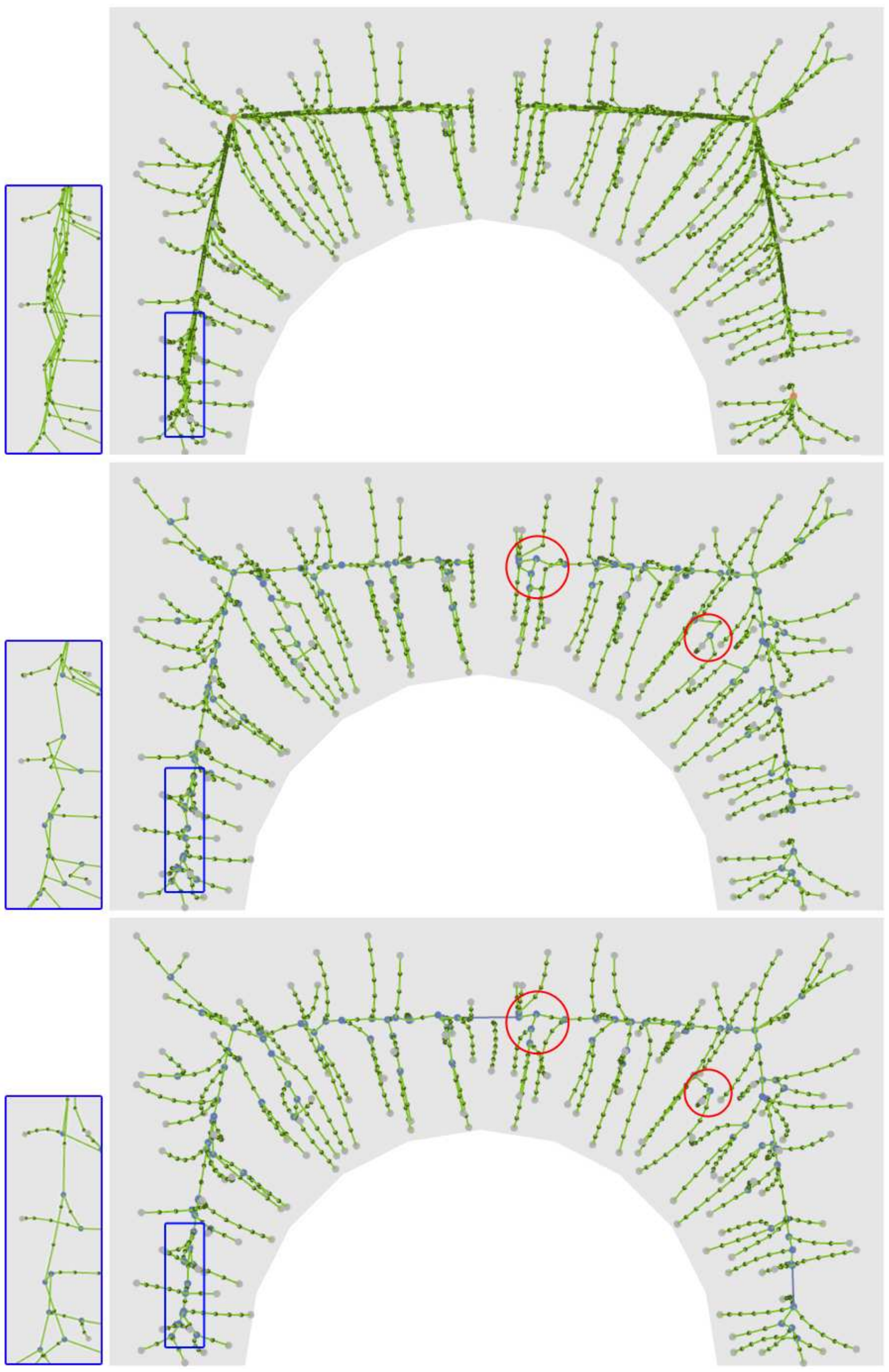
Figure 9: Top: Calculating each particle trajectory independently from each other results in clutter, since paths frequently run practically parallel to each other. Middle: The same example after merging the paths during integration which circumvents the cluttering. Bottom: The result after the components have been connected and the paths have been smoothed. The red circles show two areas where unaesthetic spikes have been removed by the smoothing algorithm.

As shown in Fig. 9 (top) these paths frequently run parallel which makes the result look cluttered and not suitable for a space frame structure. To circumvent this problem the integration stops if the current location is in proximity to an already existing position, which eventually becomes the final point of the current path. This is implemented with a simple space partitioning scheme which divides the bounding volume into small cuboids. Each time a new position p is added to the path it is inserted into the appropriate cuboid c by mapping its coordinates to indices. Then the distances between p and all other positions of different paths in c are calculated and p will be connected with the position which is closest to it. Positions on which multiple paths converge will be called crossing points henceforth. Critical points are automatically considered as crossing points regardless of the number of incident edges. The merged paths are shown in Fig. 9 (middle).

5.4. Postprocessing

Depending on the vector field, the emerging structure may consist of multiple non-connected components. These are linked to each other by connecting the two closest crossing points between them. Once a single component exists the smoothing algorithm as described in Section 4.3 is applied. Figure 9 (bottom) shows the result after the components are connected and the paths have been smoothed. Afterward cross-links — which are shown as orange slashed lines in Fig. 8 (right) — are added to the structure as follows: First, for each crossing point c and every pair of adjacent branches with vertices $(u_0 = c, u_1, \ldots, u_m)$ and $(v_0 = c, v_1, \ldots, v_n)$ a cross-link is added between u_i and v_i provided that none of the following conditions is fulfilled:

1. the angle $\alpha = \angle(\overrightarrow{u_{i-1}u_i}, \overrightarrow{v_{i-1}v_i})$ is larger than a threshold α_{max},
2. either u_i or v_i is a crossing or a boundary point,
3. the distance between u_i and v_i is larger than a maximal length l_{max},
4. a cross-link has already been added between u_i and v_i,
5. $i > n$ or $i > m$.

The process is then repeated recursively between u_{i+1} and v_{i+1}[3] until one of the above mentioned conditions is violated.

Although the basic appearance depends on the vector field, which in turn depends mostly on the geometry of the bounding volume, the results can be altered to a certain degree by changing the number and starting position of the particles as well as the parameters ϵ_d, α_{max} and l_{max} and the size of the cuboids (used for merging). Figure 2 shows a pavilion which was constructed with this algorithm.

6. Conclusions

Among the infinite number of possibilities for generating spatial structures inside a given volume, we described two methods: one is based on Voronoi tessellation and the other on repulsive force fields.

Regarding the former we currently use a uniformly distributed point cloud to generate the Voronoi tessellation. For future work, the density of the point cloud could depend on geometric properties of the boundary volume. For example, the density could be higher

[3] u_i and v_i can each only have one successor because otherwise they would either be a crossing point or a boundary point which would terminate the process.

in critical areas inside the volume, like constrictions or bottlenecks. Furthermore, different topologies can be generated by replacing the Voronoi tessellation with alternative patterns like a regular grid.

The main disadvantage of the latter method is that the resulting structures — despite randomly choosing the support points — look quite similar because the underlying vector field is defined by the bounding volume. If the vector field leads to statically unfeasible space frame structures then altering the support points will not have much effect and one cannot expect that the genetic algorithm will create a good solution. For example, a simple box only has one critical point and all particle trails will converge to this point. Therefore, the vector field should be disturbed by placing point charges randomly inside the volume which will influence the number and location of the critical points.

Acknowledgments

This work was supported by grant L358 of the Austrian Science Foundation (FWF).

References

[1] C.B. Barber, D.P. Dobkin, H. Huhdanpaa: *The quickhull algorithm for convex hulls.* ACM Trans. Math. Softw. **22**(4), 469–483 (1996). `http://www.qhull.org/`.

[2] J.L. Bentley: *Multidimensional binary search trees used for associative searching.* Commun. ACM **18**(9), 509–517 (1975).

[3] P.M. Canzarra: *Self-design and ontogenetic evolution.* In Proceedings of the Generative Art International Conference 2001.

[4] J. Chilton: *Space Grid Structures.* Architectural Press, 2000.

[5] J.-H. Chuang, C.-H. Tsai, M.-C. Ko: *Skeletonization of three-dimensional object using generalized potential field.* IEEE Trans. Pattern Anal. Mach. Intell. **22**(11), 1241–1251 (2000).

[6] N.D. Cornea, D. Silver, X. Yuan, R. Balasubramanian: *Computing hierarchical curve-skeletons of 3d objects.* The Visual Computer **21**, 945–955 (2005).

[7] T. Fischer: *Generation of apparently irregular truss structures.* Computer Aided Architectural Design Futures 2005, 229–238.

[8] A. Globus, C. Levit, T. Lasinski: *A tool for visualizing the topology of three-dimensional vector fields.* In VIS '91: Proceedings of the 2nd Conference on Visualization '91, IEEE Computer Society Press, Los Alamitos/CA, 1991, pp. 33–40.

[9] A. Hofmann, K. Bollinger, M. Grohmann: *Generating geometry of irregular frameworks algorithmically.* Proceedings of Advances in Architectural Geometry, 2008, pp. 21–23 .

[10] P.L. Jaworski: *Using simulations and artificial life algorithms to grow elements of construction.* Master's thesis, University College London, 2006. Available from `http://discovery.ucl.ac.uk/2882`.

[11] A. Kanellos: *Topological self-organisation: Using a particle-spring system simulation to generate structural space-filling lattices.* Master's thesis, University College London, 2007. Available from `http://eprints.ucl.ac.uk/4985/`.

Received August 6, 2010; final form June 23, 2011