Original scientific paper
Accepted 23. 12. 2011.

GÜNTER WALLNER
FRANZ GRUBER

Interactive Modeling and Subdivision of Flexible Equilateral Triangular Mechanisms

Interactive Modeling and Subdivision of Flexible Equilateral Triangular Mechanisms

ABSTRACT

Based on the requests from architects, we developed a system which allows to interactively design and subdivide flexible triangular surfaces. Due to economical reasons the number of different types of building elements should be small. For that reason we only use equilateral base triangles of unique size with the possibility of subdivision. To allow to interactively move vertices and to ensure constant edge length we use force-directed methods instead of inverse kinematics. This paper describes the data structure, the algorithm and the influence of subdivision on the kinematic flexibility of the mesh.

Key words: subdivision, uniform 1-to-4 split, flexible mechanism, force directed algorithm

MSC 2000: 00A67, 65D17

Interaktivno modeliranje i razdioba fleksibilnog mehanizma jednakostraničnih trokuta

SAŽETAK

Prema zahtjevima arhitekata razvijamo sustav koji dozvoljava interaktivnu tvorbu i razdiobu fleksibilne triangulirane plohe. Broj različitih sastavnih dijelova iz ekonomskih razloga treba biti mali. Zbog toga koristimo samo sukladne jednakostranične bazne trokute s mogućnošću razdiobe. Kako bismo dozvolili interaktivno kretanje vrhova i osigurali konstantnost duljine bridova umjesto inverzne kinematike koristimo metodu upravljanja silom. Rad opisuje strukturu podataka, algoritam i utjecaj razdiobe na kinematičku fleksibilnost mreže.

Ključne riječi: razdioba, uniformna 1-4 podjela, fleksibilni mahanizam, algoritam upravljanja silom

1 Introduction

In order to design the booth of the University of Applied Arts Vienna at the Vienna Fair 2011[1] (see Figure 1), the Department of Geometry was asked how to interactively model a surface consisting of equilateral triangles of different sizes. This surface, covered with sound-absorbing material (see Figure 2), was floating above the booth like a cloud and was therefore called *Acoustic Cloud* by responsible architect Juliana Herrero. For the arrangement of the triangles she required a tool which allows her to individually subdivide triangles of the mesh and to interactively move vertices while keeping the side lengths of the triangles constant. This yields a triangular mechanism whose kinematic behavior – to our knowledge – can not be simulated with existing software packages. Just as well, the theoretical solution of the task is almost impossible. Therefore, we developed a tool which attempts a numerical solution by using force-directed methods. In this paper we present the algorithm and discuss the kinematic restrictions which come along with the subdivision of the surface. Depending on the subdivision levels of adjacent triangles subdivision can enhance flexibility or not, as can be seen from the example depicted in Figure 6 and Figure 9.

Before we will describe our algorithm in Section 2 we will review related work. However, since this paper is mainly concerned with the practical application of a force-directed algorithm to a kinematic problem, we will restrict us to a short overview. Usually, force directed algorithms are associated with the drawing of graphs, where they are used to position the nodes of the graph in an aesthetically pleasing way by assigning forces to edges and nodes. Historically, the work of Tutte [9] can be considered as one of the first force-directed algorithms. Tutte only used springs of ideal length zero and no repulsive forces. Eades [3] and Fruchterman and Reingold [4] used spring forces, similar to those in Hooke's law. Both methods apply repulsive forces between all nodes and attractive forces to nodes connected by edges. However, force-directed algorithms have been applied to other domains as well.

[1] The Vienna Fair is an international contemporary art fair focusing on Central and Eastern Europe. `http://www.viennafair.at`

Figure 1: *The booth of the University of Applied Arts Vienna at the Vienna Fair 2011. (Photo by Virgil Widrich)*

Figure 2: *A close-up of the* Acoustic Cloud. *(Photo by Virgil Widrich)*

Examples include the work of Quinn and Breuer [8] who describe a force-directed method to place components on printed circuit boards. Provot [7] used a force-directed algorithm to simulate the behavior of cloth by approximating it by a deformable surface composed of a network of masses and springs. Djidjev [1] published a force-directed method to smooth unstructured triangular and tetrahedral meshes. Recently, Gruber et al. [5] described a method which optimizes given grids with rectangular topology on an arbitrary parametric double-curved surface in regard to orthogonality and, optionally, locally almost constant grid size.

2 Algorithm

2.1 Basic Data Structure

In the following we consider an initial equilateral triangular mesh $M = (V,T)$ where V is a set of vertices and T is a set of equilateral non-subdivided (Level 0) triangles with side-length a. At this point we should stress that equilaterality is not a necessity and the algorithm can be easily adapted to arbitrary triangular meshes. Additionally a set of edges E is stored. Basically, a vertex v is defined by its position in $\mathbb{R}^3$ and an edge $e = (v_0, v_1)$ connects vertex v_0 with v_1. An edge also stores references to its adjacent triangles n_0 and n_1. Note, that a triangle is only considered adjacent if both vertices v_0 and v_1 are corners of the triangle.

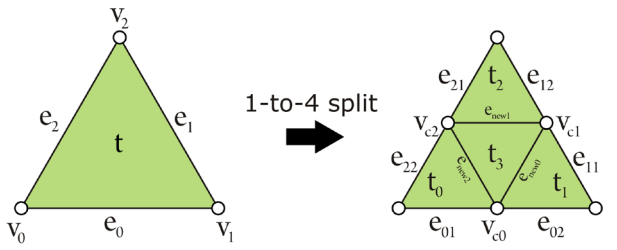

Figure 3: *Subdivision of a triangle with a 1-to-4 split.*

A triangle $t = (v_0, v_1, v_2)$ is defined by the three corners v_0, v_1 and v_2. Furthermore, references to the three edges $e_0 = (v_0, v_1)$, $e_1 = (v_1, v_2)$ and $e_2 = (v_2, v_0)$ are stored. These data structures are shown in Appendix A in Listing 1, Listing 2 and Listing 3 along with further variables which will be described later.

2.2 Subdivision

For subdivision we use a uniform 1-to-4 split, which divides a triangle t^i at subdivision level i into four triangles t_0^{i+1}, t_1^{i+1}, t_2^{i+1} and t_3^{i+1} as depicted in Figure 3. The 1-to-4 split is a commonly used method for subdivision and remeshing of triangles, e.g., Loop subdivision [6] and Butterfly subdivision [2] both use 1-to-4 refinement. To recursively traverse the hierarchy it is necessary that t^i stores pointers to these four child triangles which in turn have to store a pointer to their parent t^i. The level itself is also stored in the data structure of the triangle.

A uniform 1-to-4 split introduces new vertices v_{c0}, v_{c1} and v_{c2} at the center of the edge e_0^i, e_1^i and e_2^i of t^i. These vertices are only added to V if a vertex with the same coordinates does not already exist. We will refer to e_j^i, $j \in \{0,1,2\}$ as the *support edge* of v_{cj} and to v_{cj} as the *center vertex* of e_j^i.

Further, each edge e_j^i, $j \in \{0,1,2\}$ will be divided into two parts e_{j1}^{i+1} and e_{j2}^{i+1}. e_j^i stores references to these two child edges which in turn store a pointer to their parent e_j^i. Also, e_j^i stores a pointer to the corresponding center vertex v_{cj}. These six edges are added to E along with the three new edges e_{new0}^{i+1}, e_{new1}^{i+1} and e_{new2}^{i+1} of t_3^{i+1}. In the following we

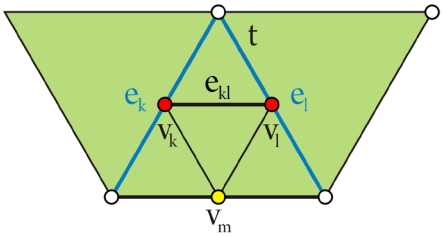

Figure 4: *Two inflexible center vertices v_k, v_l (red) of a subdivided triangle t induce a further inflexible vertex v_m (yellow).*

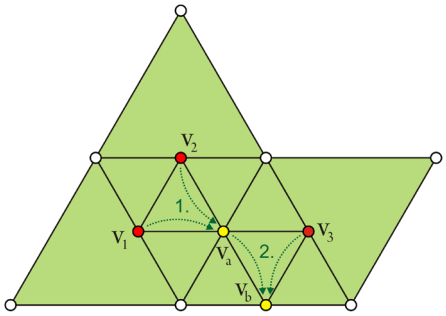

Figure 5: *An example for propagation of inflexibility: 1. the two inflexible vertices v_1 and v_2 will cause v_a to become inflexible. 2. v_a together with the already existing inflexible vertex v_3 induce a further inflexible point v_b.*

will call an edge e^0 with only one neighboring triangle (either n_0 or n_1) a *border edge*.

2.3 Kinematic Restrictions

Different subdivision levels between adjacent triangles restrict the movability of the mechanism in certain ways, i.e, just because an edge has been subdivided it does not mean that it is allowed to fold at its center vertex.

2.3.1 Inflexible Vertices

We will refer to a vertex as *flexible* if the corresponding support edge is allowed to fold, otherwise *inflexible*. All vertices which are corners of a triangle at level 0 are flexible. The flexibility of vertices v_{cj}, $j \in \{0,1,2\}$ introduced during subdivision is algorithmically determined by performing the following test. If there already exists a vertex v^* with the same coordinates then v^* is marked as flexible. Otherwise, we check if the corresponding support edge e_j is a border edge. If it is, v_{cj} is flexible, if it is not v_{cj} is inflexible. In other words, a center vertex v with support edge e is inflexible if and only if one of the adjacent triangles of e is subdivided, otherwise v is flexible.

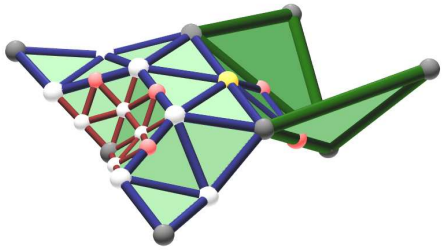

Figure 6: *An example of a 3D mechanism. Different subdivision levels result in inflexible vertices (shown in red and yellow). Such vertices prevent folding of their support edge and therefore restrict the movability of the mechanism in regard to interactive displacement.*

2.3.2 Induced Inflexible Vertices

However, the test in Section does not capture all kinematic restrictions because it does not take into account the configuration of the neighborhood. Certain configurations can cause a propagation of inflexibility through the mesh, in other words, inflexible vertices can induce further inflexible vertices what we therefore call *induced inflexible vertices* in the following. To detect these induced inflexible vertices we run a second test after the above mentioned test.

If a triangle t is subdivided and two of the three center vertices v_k and v_l of its edges are inflexible (as shown in Figure 4) then the third center vertex v_m also becomes inflexible because of the following reason. If v_k is inflexible then the corresponding support edge e_k is not allowed to fold, the same holds true for v_l and its support edge e_l. Therefore the edge e_{kl} between v_k and v_l is also rigid, even if the sub-triangles of t are further subdivided. e_k and e_l together with e_{kl} as a distance piece between them make the triangle t rigid.

The introduction of an induced inflexible vertex can lead to further induced inflexible vertices. A simple example is shown in Figure 5. To handle this possible series of reactions, the test has to be repeated until no further induced inflexible vertices are found. Figure 6 shows a 3D example of a surface with different subdivision levels and the different types of inflexible vertices.

2.4 Force-directed Placement

An important feature of the tool is that the surface can be interactively modeled by moving vertices. Since a theoretical calculation with inverse kinematic of such a mechanism is almost impossible, we attempt a numerical solution with force-directed methods. The iterative force-directed algorithm is responsible for arranging the vertices in a way that

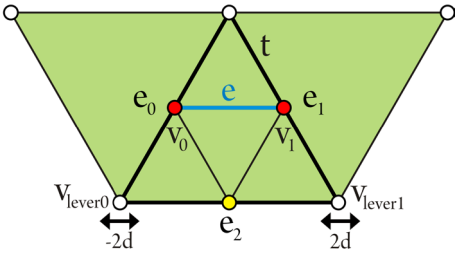

Figure 7: *In order to keep the distance between two inflexible vertices v_0 and v_1 constant, their displacements have to be applied twice to the corresponding vertices v_{lever0} and v_{lever1} instead to v_0 and v_1.*

guarantees that edges keep their predefined ideal length. In case of equilateral triangles the $idealLength = 0.5^i \cdot a$, where i is the subdivision level of the edge and a is the side length of a level 0 triangle. However, the set of edges E contains more information than necessary for the simulation. Therefore, we derive a minimal set of edges $F \subseteq E$ which completely describes the flexible mechanism.

To determine the edges of F, we initially loop only through edges e which do not have a parent[2]. In order to decide which parts of the edge hierarchy of e belong to F, we start the following recursive process. If e does not have any children, the edge itself belongs to F. Otherwise, we check if the center vertex of e is inflexible in which case e is also part of F. However, if the center vertex is flexible then the process is repeated recursively for $e = e_{c0}$ and $e = e_{c1}$ where e_{c0} and e_{c1} are the two child edges of e.

For each vertex v a displacement vector v^{disp} is stored which is set to zero at the beginning of each iteration. Afterward, the algorithm loops through each edge $e = (v_0, v_1) \in F$ and calculates the deviation Δ from the actual edge length $\Delta = e_{ideal} - \|v_{01}\|$ where $v_{01} = v_1^{pos} - v_0^{pos}$. If v_0 is flexible then $d := \varepsilon \cdot \Delta \cdot \frac{v_{01}}{\|v_{01}\|}$ is subtracted from v_0^{disp}, where $\varepsilon > 0$ is a small constant. Similarly d is added to v_1^{disp} if v_1 is flexible.

No displacement is added to inflexible points because their position has to remain at the center of their rigid support edge. This calculation will be performed later. However, if both vertices of an edge are inflexible, as depicted in Figure 7, a problem arises because the edge would not have the effect of a distance piece anymore. We solved the problem by transferring the edges' distance keeping force onto other suitable vertices. Let e_0 be the support edge of v_0, e_1 the support edge of v_1 and t the triangle which contains e_0 and e_1. Further, let e_2 be the third edge of t. Then the displacements caused by e will be applied twice to the corresponding vertices v_{lever0} and v_{lever1} of e_2.

[2]Note, that this are not only edges on subdivision level 0.

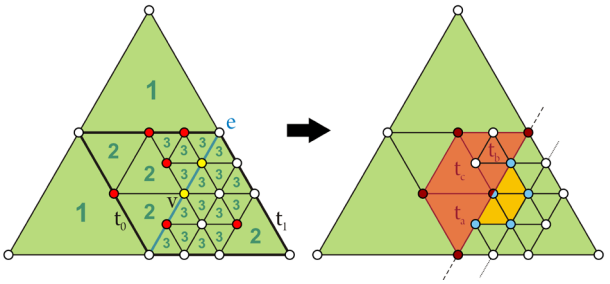

Figure 8: *An example for the planarity condition of an (induced) inflexible vertex v with support edge e and adjacent triangles t_0 and t_1: The left image shows the subdivision level of triangles and the flexibility of vertices. The right image shows the trapezoids implied by v for t_0 (red) and t_1 (orange). The corners of each separate trapezoid together with v have to be coplanar. The different sizes of the trapezoids are a result from the different subdivisions at both sides of e. The smaller orange trapezoid allows folding the mesh at the dotted line, which would not be the case if the three adjacent level 2 triangles would have been chosen for the trapezoid.*

Once all edges have been processed the aggregated displacement of each vertex is added to its position. Finally, the calculation of the position of inflexible vertices is performed. This calculation has to be carried out in a hierarchical top to down manner to make sure that the center vertex of the parent of an edge f is already at the right position before calculating the position of a possible center vertex of f.

These steps are summarized in Listing 4 in Appendix A.

2.4.1 Planarity Condition and Convergence Speed

Because the algorithm described above only takes edge lengths and not planarity into account it will converge slowly to the solution. However, an (induced) inflexible vertex will not only make its support edge rigid but will also make parts of the surface planar. We take advantage of this theoretical fact to improve the converge speed of the algorithm. Let us assume that v is an (induced) inflexible vertex with e being its support edge and let t_0^i and t_1^i be the adjacent triangles of e at level i. In the following let us consider t_0^i to be subdivided (the same process is performed for t_1^i if it is subdivided). Figure 8 shows a possible configuration. Inside the hierarchy of t_0^i we search for three triangles incident to v with the following properties:

1. All three triangles are on the same subdivision level m
2. There is no higher level $k > m$ which fulfills condition (1), i.e., m is maximal

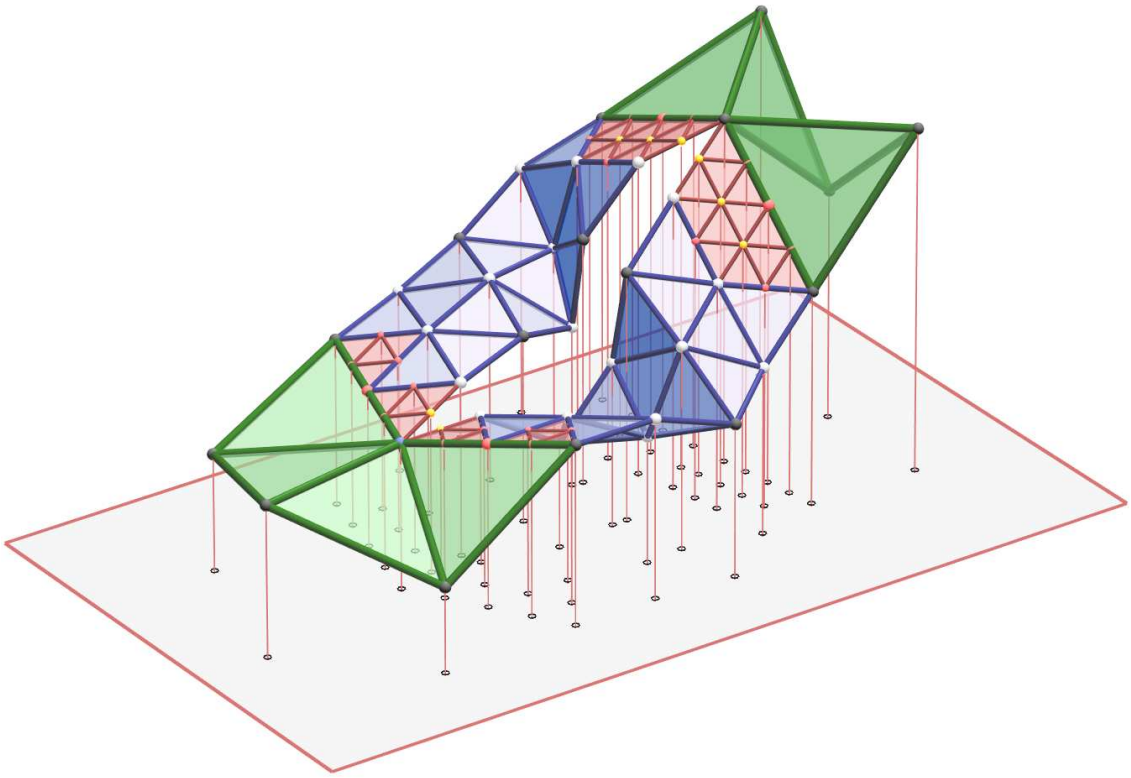

Figure 9: *An example of a mesh with subdivisions up to level 2 which was designed with the described tool (level 0 = green, level 1 = blue and level 2 = red). Whereas triangles on subdivision level 0 and 1 have high flexibility, the level 2 triangles – as chosen in this example – have no further influence on the kinematic behavior of the mechanism. Inflexible vertices are shown in red and induced inflexible vertices are colored yellow.*

All vertices of these three triangles have to be coplanar, because the support edge e of v can be thought of as a frame joint for the two triangles t_a and t_b as seen in Figure 8. Together with the remaining triangle t_c they form an isosceles trapezoid. v stores these vertices in an array called trapezoidVertices0 in the data structure. After displacement of the vertices in the force-directed algorithm, the vertices of this array are orthogonally projected onto their regression plane. This procedure increases the speed of the algorithm considerably.

2.4.2 Termination Condition

The steps outlined above are repeated iteratively until all displacements are below a certain threshold, mathematically $v_{disp} < \varepsilon \quad \forall v \in V$.

3 Operations

An important aspect in the development of the system was interactivity. Users should be able to easily work with the mesh in order to facilitate the design process. Therefore all operations can be performed with a single mouse click together with keyboard shortcuts.

The system allows to append edges of side length a to corners of level 0 triangles. If adding a new edge yields a new base triangle (i.e., a triangle at subdivision level 0) then this triangle is automatically added to T. Only edges with no adjacent triangles can be removed. Existing triangles – independent of their subdivision level – can be subdivided with a 1-to-4 split, as described in Section 2.2. When deleting a triangle two cases can occur: If a triangle t^i with $i = 0$ (base triangle) is deleted then edges not adjacent to any remaining triangles are deleted automatically as well. In case $i > 0$ all four triangles of the parent triangle from t^i, including edges and vertices which are no longer part of a remaining triangle, are deleted in order to guarantee a well-defined subdivision hierarchy. Dragging a vertex with the left (right) mouse button moves it horizontally (vertically). Once the position of a vertex changes the arrangement of the mesh has to be recalculated which is handled by the force-directed algorithm described in Section 2.4.

4 Conclusions

In this paper we described a tool to interactively design triangular mechanisms by extending the mesh, subdividing triangles, and moving vertices (Figure 9 shows an example). Although we restricted ourselves – due to the initial artistic concept – to equilateral triangles it should be noted that the algorithm can be easily modified to general triangles.

To simulate the kinematic behavior of the mesh we used an iterative force directed algorithm. As it turned out, subdivisions must not necessarily increase the flexibility, because in many cases inflexibility propagates quite easily. This means that subdivision has to be applied well-considered in order to raise flexibility. From a designer's point of view, this may be very restrictive. In general the flexibility can be improved if one allows small deviations from the ideal edge length. Experiments showed that ignoring the planarity condition (see Section 2.4.1) and allowing for the edge length to deviate only by 1% from its predefined ideal length, increases the flexibility already considerably.

A Data Structures and Pseudo Code

```
struct sVertex
{
  float[3] position;
  float[3] displacement;

  bool flexible;
  bool inducedInflexible;

  sEdge *supportEdge;

  Array<sVertex*> trapezoidVertices0;
  Array<sVertex*> trapezoidVertices1;
};
```

Listing 1: Vertex data structure

```
struct sEdge
{
  sVertex *v0;
  sVertex *v1;
  sVertex *center;

  sEdge *c0;  // child 0
  sEdge *c1;  // child 1
  sEdge *parent;

  sTriangle* n0;
  sTriangle* n1;

  float idealLength;
  int level;
};
```

Listing 2: Edge data structure

```
struct sTriangle
{
  sVertex* v0;
  sVertex* v1;
  sVertex* v2;

  sEdge* e0;
  sEdge* e1;
  sEdge* e2;

  sTriangle* c0;  // child 0
  sTriangle* c1;  // child 1
  sTriangle* c2;  // child 2
  sTriangle* c3;  // child 3
  sTriangle* parent;

  int level;
};
```

Listing 3: Triangle data structure

```
foreach (sVertex v)
  v.displacement = (0,0,0);

foreach (sEdge e) {
  float[3] l = e.v0.position - e.v1.position;
  float Δ = e.idealLength - l.length();

  l.normalize();

  if (e.v0.flexible == true)
    e.v0.displacement += -ε*Δ*l;
  if (e.v1.flexible == true)
    e.v1.displacement +=  ε*Δ*l;

  if (e.v0.flexible==false &&
      e.v1.flexible==false) {
    // find lever0 and lever1, see Fig. 6
    lever0.displacement += -2*ε*Δ*l;
    lever1.displacement +=  2*ε*Δ*l;
  }
}

foreach (sVertex v)
  v.position += v.displacement;

calcPositionOfInflexiblePoints();

foreach (trapezoid t) {
  Plane regPlane = calcRegressionPlane(t);
  foreach (Vertex v in t.vertices)
    regPlane.orthoProject(v);
}
```

Listing 4: Pseudocode for the force-directed algorithm from Section 2.4

References

[1] H. DJIDJEV, Force-Directed Methods For Smoothing Unstructured Triangular And Tetrahedral Meshes. *Proceedings of the 9th International Meshing Roundtable* (2000), 395–406

[2] N. DYN, J. GREGORY, D. LEVIN. A BUTTER, A Butterfly Subdivision Scheme for Surface Interpolation with Tension Control. *ACM Trans. Graph.* **9** (1990), 160–169

[3] P. A. EADES, A heuristic for graph drawing. *Congressus Numerantium* **42** (1984), 149–160

[4] T. M. J. FRUCHTERMAN, E. M. REINGOLD, Graph Drawing by Force-directed Placement. *Software - Practice and Experience* **21** (1991), 1129–1164

[5] F. GRUBER, G. WALLNER, G. GLAESER, Force Directed Near-Orthogonal Grid Generation on Surfaces. *Journal for Geometry and Graphics* **14** (2010), 135–145

[6] C. LOOP., Smooth subdivision surfaces based on triangles. *Masters Thesis*, Utah University (1987)

[7] X. PROVOT, Deformation constraints in a mass-spring model to describe rigid cloth behavior. *Graphics Interface* (1995), 147–154

[8] N. QUINN, M. BREUER, A force directed component placement procedure for printed circuit boards. *IEEE Transactions on Circuits and Systems* (1979), 377–388

[9] W. T. TUTTE, How to draw a graph. *Proceedings of The London Mathematical Society* **13** (1963), 743–767

Günter Wallner
e-mail: guenter.wallner@uni-ak.ac.at

Franz Gruber
e-mail: franz.gruber@uni-ak.ac.at

University of Applied Arts Vienna
Department of Geometry
1010 Vienna, Austria