# Connectivity Shapes

Martin Isenburg*
University of North Carolina
at Chapel Hill

Stefan Gumhold†
University of Tübingen

Craig Gotsman‡
Technion – Israel Institute
of Technology

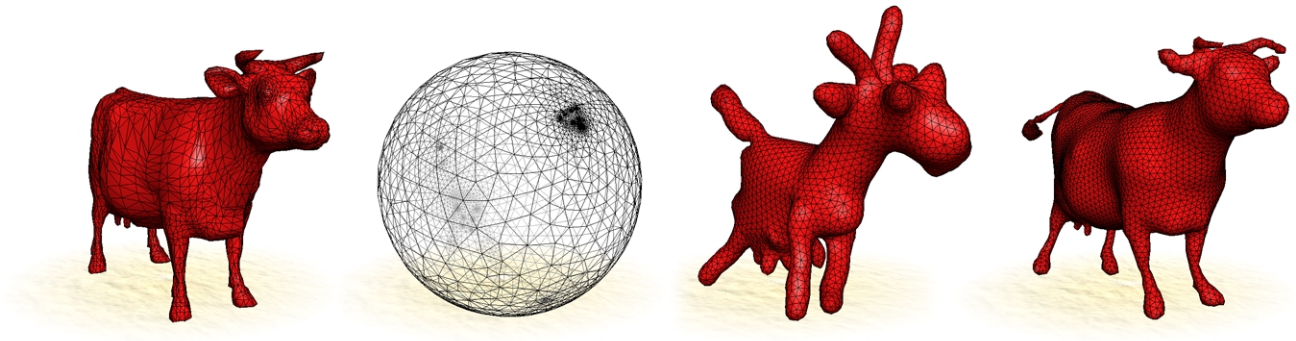

Figure 1: **(a)** Original polygonal mesh of a cow. **(b)** The connectivity of (a) embedded on the sphere. **(c)** The connectivity shape corresponding to the connectivities of (a) and (b). **(d)** Connectivity shape after remeshing the cow of (a).

## Abstract

We describe a method to visualize the connectivity graph of a mesh using a natural embedding in 3D space. This uses a 3D shape representation that is based solely on mesh connectivity – the *connectivity shape*. Given a connectivity, we define its natural geometry as a smooth embedding in space with uniform edge lengths and describe efficient techniques to compute it. Our main contribution is to demonstrate that a surprising amount of geometric information is implicit in the connectivity.

We also show how to generate connectivity shapes that approximate given 3D shapes. Potential applications of connectivity shapes to modeling and mesh coding are described.

**Keywords:** Natural embedding, mesh connectivity, implicit geometry, polygon meshes, shape compression.

## 1 INTRODUCTION

The most widely used representation for three-dimensional geometric surfaces are polygonal meshes. These consist of mesh *geometry* and mesh *connectivity*, the first describing the location of each vertex in 3D space and the latter describing how to connect the vertices together to form polygons that describe a surface.

At first glance, these two components of the mesh seem to be independent, namely that many different mesh connectivities could co-exist with a given geometry, and vice versa. While this is theoretically true, it also seems that, in typical real-world meshes, some correlation between the two exists. For example, ”nice” triangulations of mesh geometries, such as the Delaunay-type triangulations, are traditionally preferred over ”ugly” triangulations containing long and skinny triangles.

This form of the correlation imposes a ”natural” connectivity on a given geometry, and has been investigated in the context of mesh generation from 3D point clouds in 3D scanning applications (e.g. [1]) and optimal triangulations of point sets (e.g. [2]). The reverse, imposing a ”natural” geometry on a given connectivity, has been treated less. This paper explores mainly that direction.

We introduce a shape representation that is based solely on connectivity. In Figure 1(a) we see a well-known polygonal model of a cow. Ignoring the geometry, we have mapped the cow’s connectivity onto the unit sphere (b), where the different densities hint to the features of the cow. In (c) the corresponding *connectivity shape* is shown. It is a smooth embedding with uniform edge lengths of the connectivity graph of (a) and (b) in three dimensional space.

Imagine all edges of the cow being springs of the same equilibrium length. In the embedding (b) we forced the spring system into a high energy state. In (c) we released all vertices and the spring system relaxed into a low energy state, with more or less uniform edge lengths. This can be thought of as the connectivity’s natural shape. More poetically, the sphere embedding in (b) has the *body* of a sphere, but the *soul* of an animal. The embedding in (c) reveals the geometric *soul* of the cow’s connectivity.

Connectivity shapes are closest in spirit to the embedding techniques used in the graph drawing community. The focus of most of their attention has been on 2D embedding methods, and only recently have embeddings in 3D [22, 10, 7] become popular for visualization purposes. These graph drawing approaches use re-

*isenburg@cs.unc.edu http://www.cs.unc.edu/~isenburg/cs
†stefan@gumhold.de
‡gotsman@cs.technion.ac.il

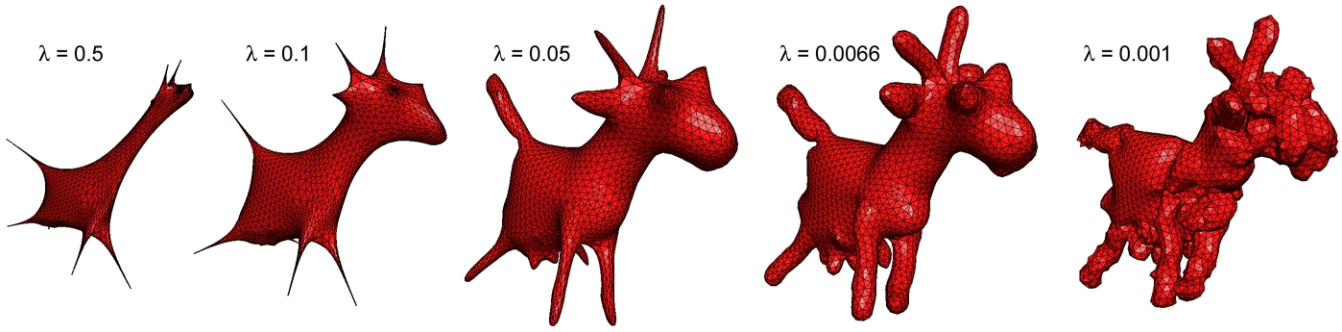


Figure 2: Family of connectivity shapes computed from the cow's connectivity with different smoothing parameters $\lambda$.

pelling force techniques that allow to embed graphs of arbitrary topology. However, for graphs that have the topology of a surface (e.g. that are 2-manifold), the connectivity shape technique will produce more aesthetically pleasing results, especially when the graph's surface characteristics are an important feature of the visualization.

We can also generate connectivities whose natural shape resembles that of a *given* shape. This is done by (re)meshing the given shape with uniform edge lengths. For example, the connectivity shape in Figure 1(d) bears a striking resemblance to the original (a). The only information in this mesh is its connectivity, in the sense that it induces the mesh geometry.

In the following section we define a connectivity shape as the natural geometry associated with a connectivity and describe a method to compute it. In Section 3 we show how to create connectivities whose connectivity shape approximates a desired shape. Hierarchical methods for faster computation of connectivity shapes are the topic of Section 4. Finally we summarize our work, describe some open theoretical problems and discuss potential applications of connectivity shapes, such as modeling and mesh coding.

## 2 SHAPE FROM CONNECTIVITY

Any polygon mesh can be thought of as an embedding of a connectivity graph in three dimensions. The location of each vertex of the mesh is carefully chosen since the surface is meant to represent a geometric shape. However, our thesis is that for most meshes there is a substantial amount of information about the shape present in its connectivity. To illustrate, consider Figure 1. In (b) the vertices of the connectivity graph of the cow mesh in (a) have been embedded on the unit sphere such that each vertex is approximately at the center of its neighbors. This embedding is computed from the connectivity alone without any knowledge about the geometry. However, the geometric features of the cow are reflected in the vertex densities of the embedding. The regions of the connectivity graph covering the body of the cow map uniformly onto the sphere, whereas prominent extremities such as the legs or the tail map to dense concentrations, suggesting that they would rather "pop" out of the sphere and must be forced together by the mapping.

We now define a preferred geometric shape based on the connectivity alone. Given a connectivity graph $\mathcal{C} = (\mathcal{V}, \mathcal{E})$, consisting of an indexed set of $n$ vertices $\mathcal{V} = \{v_i\}_{i=1\ldots n}$ and a set of $m$ undirected edges $\mathcal{E} = \{e_j = (i_1(j), i_2(j))\}_{j=1\ldots m}$, the *connectivity shape* $CS(\mathcal{C})$ corresponding to $\mathcal{C}$ is a list of $n$ vectors $x = (x_i \in \mathbb{R}^3)_{i=1\ldots n}$ associated with the graph vertices that best satisfy some natural property. We have chosen the property that all edges have unit length. This choice corresponds to the intuition that the connectivity shape can be interpreted as the equilibrium state of a system of springs with identical equilibrium length joined together according to the connectivity graph. Alternatively, the connectivity shape can be thought of as an isometric embedding of the connectivity graph in $\mathbb{R}^3$, which is a solution to the following set of quadratic equations in the $3n$ unknowns:

$$||x_i - x_j||^2 = 1 \qquad \forall (i,j) \in \mathcal{E} \tag{1}$$

The number of equations, $m$, is the number of edges in $\mathcal{E}$ and determined by the Euler-Poincaré formula for a shell of genus $g$ with $n$ vertices and $f$ faces: $m = n + f + 2g - 2$.

On one hand the quadratic nature of the equations (1) introduces ambiguity into the solution. This can be seen by considering a vertex with a circular symmetric neighborhood forming a cap protruding *out* of the shape. Then the same shape with the vertex protruding *into* the shape is an equivalent solution to the system; it cannot distinguish between the two. On the other hand the quadratic nature might not admit a solution at all. Hence, we seek a solution in the least squares sense by minimizing the following *spring energy*:

$$E_S\left(x \in \mathbb{R}^{n\times 3}\right) = \sum_{(i,j)\in\mathcal{E}} \left(||x_i - x_j|| - 1\right)^2 \tag{2}$$

Furthermore, the problem of an isometric embedding of a two dimensional manifold in three dimensions has no smooth solution in general. The geometry minimizing (2) typically does not have a "nice" shape, rather it will be a rough surface with extremely high local curvatures. In order to bound the curvatures of the solution and also eliminate unwanted local minima resulting from the in/out cap protrusion problem, it is necessary to *regularize* the solution by adding a roughness term to the cost function. This term $E_R$ is defined using the discrete Laplacian operator

$$\mathcal{L}(x_i) = \frac{1}{d_i} \sum_{(i,j)\in\mathcal{E}} x_j - x_i \tag{3}$$

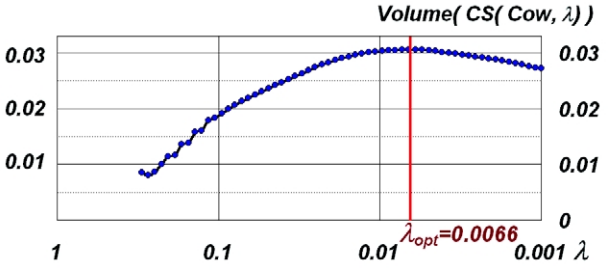


Figure 3: Finding the optimal smoothing parameter $\lambda$: The volume of the connectivity shapes from Figure 2 as a function of $\lambda$.

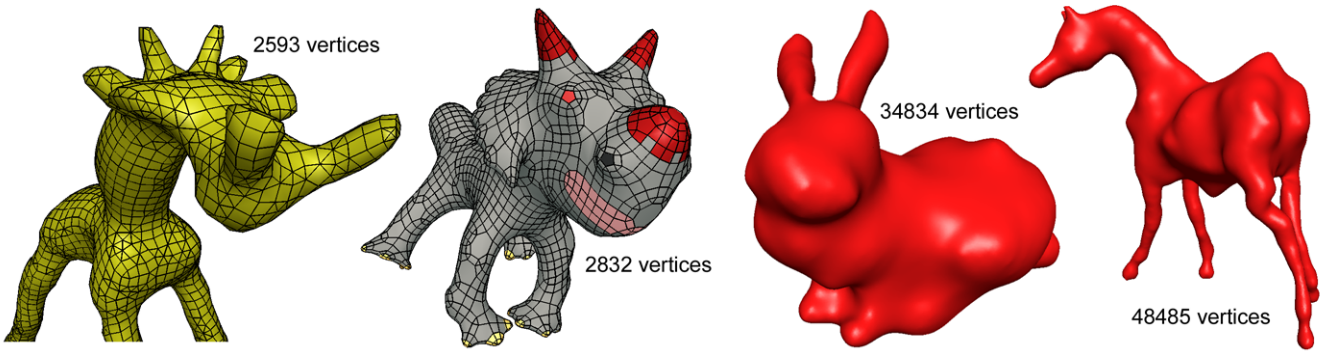


Figure 4: Connectivity shapes computed from the connectivity of (a) giraffe, (b) triceratops, (c) Stanford bunny, and (d) horse.

where the sum is over all edges incident on node $i$ and $d_i$ is the degree of node $i$. Minimization of the roughness energy term

$$E_R\left(x \in \mathbb{R}^{n\times 3}\right) = \sum_{i=1}^{n} \mathcal{L}(x_i)^2 \tag{4}$$

also minimizes the second derivatives and thus the curvature of the shape. Using the two energy terms $E_S$ and $E_R$ we define a family of connectivity shapes

$$CS(\mathcal{C},\lambda) = \underset{x\in R^{n\times 3}}{\operatorname{argmin}} \underbrace{\left[(1-\lambda)E_S(x) + \lambda E_R(x)\right]}_{E_\lambda(x)} \tag{5}$$

The tradeoff between unit edge length and smoothness is controlled by the positive real parameter $\lambda$. Figure 2 illustrates the influence of this parameter on the resulting connectivity shapes. Large values of $\lambda$ result in very skinny and smooth meshes. Small values of $\lambda$ cause the mesh to inflate to its natural shape until it becomes bumpy and looses its appeal. The obvious question is how to choose the optimal value $\lambda_{opt}$. It should be as small as possible without introducing bumpiness. A human observer can easily find the optimal smoothing parameter by visual inspection, but an automatic method is more elusive. We have found that the volume of the mesh is useful in this context:

$$\lambda_{\mathrm{opt}}(\mathcal{C}) = \underset{\lambda\in[0,1]}{\operatorname{argmax}} \; \mathit{Volume}\left(CS(\mathcal{C},\lambda)\right) \tag{6}$$

namely, the optimal smoothing parameter $\lambda_{opt}$ corresponds to the value $\lambda$ at which the connectivity shape has maximal volume.

Figure 3 plots the volume of the connectivity shapes of Figure 2 as a function of $\lambda$. The volume is small for large values of $\lambda$ and increases for smaller values until it reaches the maximum volume at $\lambda_{opt}$. When the shape starts to become bumpy the volume decreases again. This happens because the total surface area of the mesh is fixed by the unit edge lengths and volume decreases with increasing fractality of the surface.

Figure 4 shows various examples of connectivity shapes generated with optimal smoothing parameters for the connectivities of some popular polygonal meshes. Remember, that these shapes do not use any of the original geometric information, but are computed from the connectivities of the meshes alone.

## 2.1 Embedding with an Iterative Solver

Generating a connectivity shape implies a numerical solution to the optimization problem (5). Since the input is just a connectivity graph, and we use an iterative method, we have to compute an initial location for each vertex. For polygon meshes with genus zero we use an embedding of the mesh on the unit sphere. While a pure 3D approach to this is difficult, an easy way out is to reduce it to two 2D problems. This is achieved by finding a *vertex separator* that partitions the connectivity graph into two components with approximately the same number of vertices in each component and a common boundary. Each component may be mapped to the unit disk by the Tutte procedure [21], which positions the boundary vertices at fixed locations on a circle, and each interior vertex at the centroid of its neighbors. We use a variant of the Tutte procedure which computes positions on half a sphere, rather than in the plane. Thanks to the common boundary, the two components fit together perfectly along the sphere equator.

The iterative solver then proceeds to solve (5) using the conjugate gradient method. A fast conjugate gradient solver relies on two major components. Firstly, a method to compute the gradient of the energy $\nabla E_\lambda(x) \in \mathbb{R}^{n\times 3}$, where $\nabla$ consists of all the partial derivatives $\partial/\partial x_{1,1}, \partial/\partial x_{1,2}, \partial/\partial x_{1,3}, \partial/\partial x_{2,1}, \ldots, \partial/\partial x_{n,3}$. The second component of the solver computes for a given displacement vector $d \in \mathbb{R}^{n\times 3}$, added to the current solution $x$, the real parameter $\alpha_{opt}$, such that

$$\alpha_{\mathrm{opt}} = \underset{\alpha>0}{\operatorname{argmin}} \; E_\lambda(x+\alpha\cdot d).$$

In the steepest descent method just the negative energy gradient is used as the displacement vector $d$. In the conjugate gradient method $d$ is incrementally calculated from the negative gradients in a way that avoids unnecessary iterations in directions previously explored. The calculation of the step size $\alpha_{\mathrm{opt}}$ can accelerate the convergence of the conjugate gradient method significantly. In our case $\alpha_{\mathrm{opt}}$ cannot be expressed analytically, since $E_S$ is not polynomial in $x$. To overcome this, we use a modified version of the spring energy

$$E_S'\left(x \in \mathbb{R}^{n\times 3}\right) = \sum_{(i,j)\in\mathcal{E}} \left(||x_i - x_j||^2 - 1\right)^2, \tag{7}$$

where the edge lengths have been squared. $E_S'$ has the same global minimum as $E_S$ in the case there exists a smooth unit edge length embedding. The energy $E_\lambda' = E_S' + E_R$ is a quartic polynomial in $x$, so $\alpha_{\mathrm{opt}}$ can be computed analytically. After minimization of $E_\lambda'$ the solver continues to minimize $E_\lambda$ with only a few iterations.

A second method of accelerating convergence, which also avoids most local minima, is to *cool* down the smoothing. The initial spherical embedding is very smooth and corresponds to $\lambda = 1$ (under the constraint that all vertices are on the sphere). From there we continue with $\lambda = 0.5$. After convergence we halve $\lambda$ and continue again. This terminates when the target smoothing parameter is reached. The discrete halving is preferred over a continuous decrease of $\lambda$ because it reduces the number of restarts of the conjugate gradient solver. Excessive restarting would cause our solver to degenerate into the slower steepest descent solver.

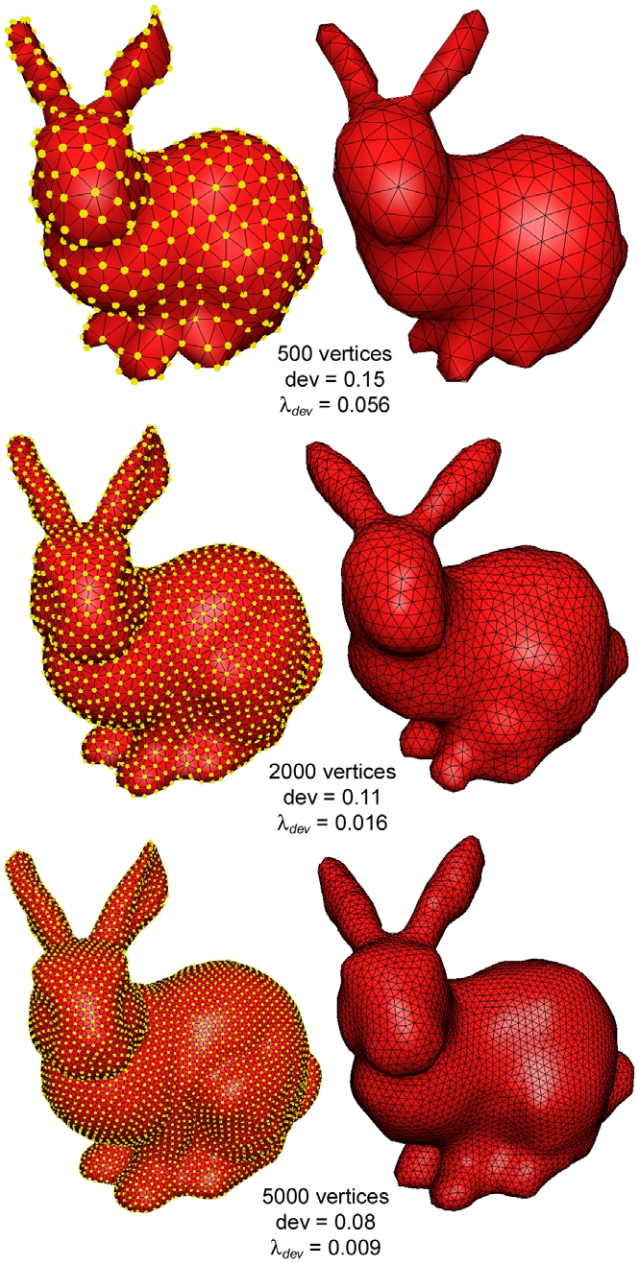


Figure 5: Three bunnies remeshed with 500, 2000, and 5000 vertices (left) and their corresponding connectivity shapes (right). Reported are the relative deviation in edge length $\mathrm{dev}$ after remeshing and the smoothing parameter $\lambda_{\mathrm{dev}}$.

The convergence of various connectivity shapes is illustrated in the accompanying video. For large meshes the convergence is quite slow. Hierarchical methods (see Section 4) can speed it up.

# 3 CONNECTIVITY FROM SHAPE

So far we have created the natural geometry for a given connectivity. In this section we describe how to do the opposite, namely how to generate a connectivity graph whose corresponding connectivity shape approximates the geometry of a given mesh.

Connectivity shapes aim, by definition, at achieving unit edge length for the entire mesh. This is the main reason that the connectivity shapes of the giraffe (a) and the triceratops (b) in Figure 4 look quite different from the original polygon meshes. These meshes were created with a modeling package and their edge lengths are non-uniform. On the other hand, the connectivity shapes of the Stanford bunny (c) and the horse (d) better resemble the original meshes. These meshes were created by surface reconstruction from the point cloud produced by a 3D scanner. Due to the regular spacing of the sample points, they have fairly uniform edge lengths.

Had we used the original edge lengths instead of a unit edge length of 1 in Eq. (7), the resulting shapes would always closely resemble the original. But we aim at a shape whose geometry is a function of connectivity *only*. To generate a connectivity whose natural shape matches a given shape, we need to create a triangle mesh with edges of equal length that describes this shape. Figure 6 shows examples of connectivity shapes approximating some popular polygon meshes. We generated these connectivities using the method described in the next subsection.

## 3.1 Meshing and Remeshing

The process of creating a polygonal mesh from a geometric shape is called *meshing*. If the geometric shape is already in a polygonal form, generating another polygonal mesh is also called *remeshing*. Our objective is a (re)meshing method that results in a faithful approximation of a given shape while using only edges of unit length. In the ideal case the resulting geometry will then be a solution of (5) given the resulting connectivity.

Mesh generation techniques used in the finite-element community have a similar objective. For fast convergence and high accuracy of numerical computations they require meshes with optimal *element shapes*, which usually means equilateral triangles. For the 2D case, a wealth of meshing methods exist (e.g. see survey in [2]). The 3D surface case has been treated less, but some notable methods are those of Turk [20] and Frey [6].

It seems quite difficult to optimize both the geometry and the connectivity of the mesh such that on the one hand unit edge lengths are achieved, and on the other hand we remain faithful to the original shape. Instead of attempting this, we customized Turk's retiling procedure [20], which was well suited for our purposes.

Turk's method distributes $n$ points as uniformly as possible over the mesh and triangulates them into a surface. The variation in edge length is then further minimized with local edge flips. Uniform distribution of the points is achieved by applying a relaxation method to initially randomly placed points on the mesh. This method associates with each point a force that repels neighboring points within a radius $r$. This radius reflects the target edge length in the vicinity of the point. For uniform edge length this is a constant that only depends on the total surface area and the number of points $n$.

Our variation alternates between two stages: One relaxes the positions of the points with Turk's method and the other displaces them by Laplacian smoothing. The connectivity between the points used for the Laplacian smoothing is constantly re-computed, aiming at uniform edge length. While these two stages alternate, we slowly decrease the amount of of relaxation and smoothing until the points are *frozen*. Remeshed versions of the Stanford bunny using 500, 2000, and 5000 vertices and the connectivity shapes computed from their connectivities are shown in Figure 5.

Remeshing with a larger number of vertices not only results in a more accurate approximation of the original shape, but also reduces the spring energy (2), as the relative deviation (the ratio between the standard deviation and average) in edge length decreases—both factors bring the connectivity shape closer to the desired shape. Increasing the number of vertices, however, increases the complexity of the generated connectivity, which makes the computation of its connectivity shape more expensive.

From the relative deviation in edge length of the remeshed shape we can determine a smoothing parameter $\lambda_{\mathrm{dev}}$ for which the con-

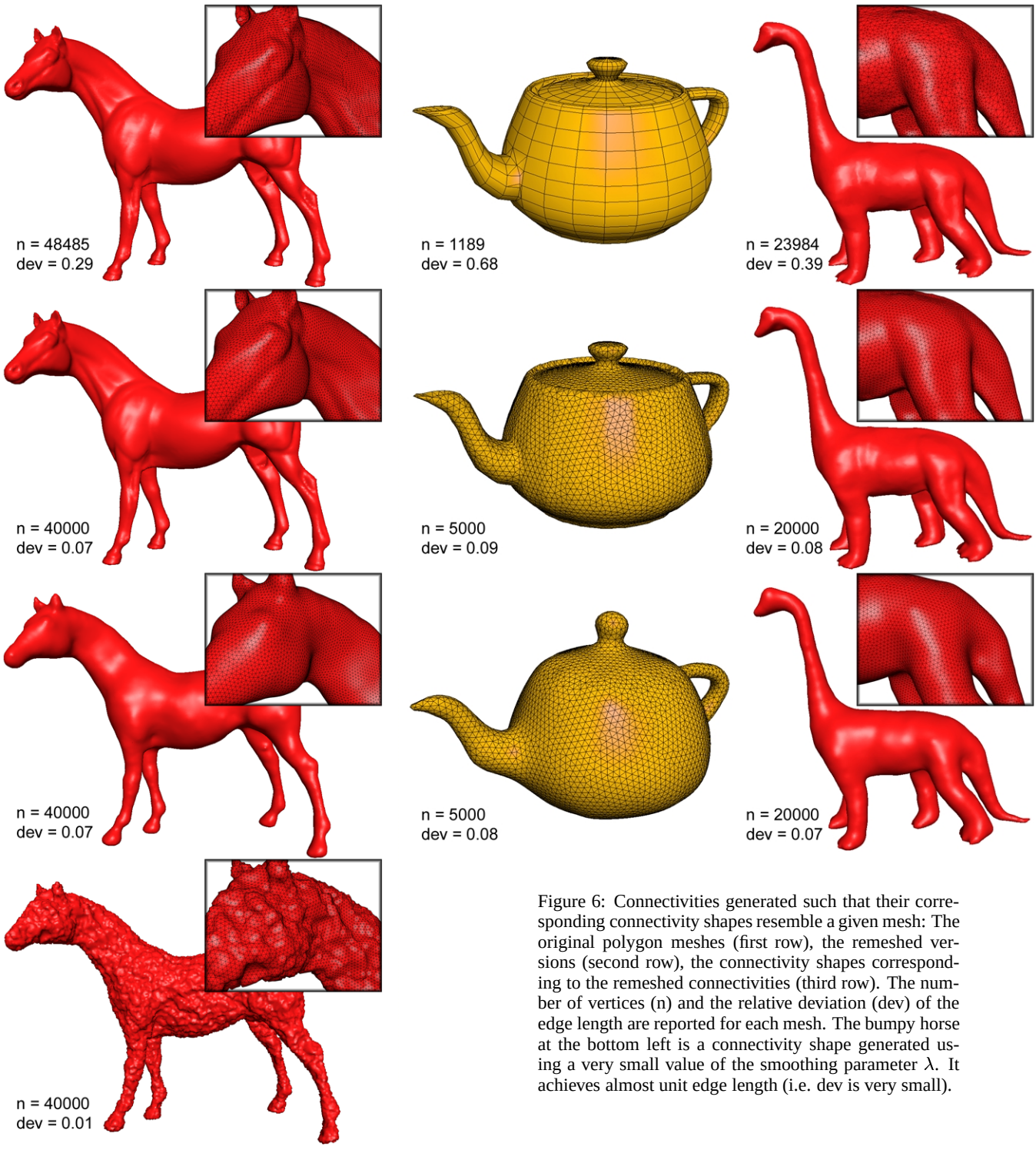


Figure 6: Connectivities generated such that their corresponding connectivity shapes resemble a given mesh: The original polygon meshes (first row), the remeshed versions (second row), the connectivity shapes corresponding to the remeshed connectivities (third row). The number of vertices (n) and the relative deviation (dev) of the edge length are reported for each mesh. The bumpy horse at the bottom left is a connectivity shape generated using a very small value of the smoothing parameter $\lambda$. It achieves almost unit edge length (i.e. dev is very small).

nectivity shape will be a good match. Connectivity shapes try to realize unit edge length for the entire mesh (i.e. relative deviation of zero). In general the edges of the remeshed shape will not have unit length, so we can choose the smoothing parameter at which the connectivity shape has the same relative deviation in edge length as the shape it is trying to match. Experimentally this $\lambda_{\mathrm{dev}}$ turns out to be close to the optimal $\lambda_{opt}$, as defined in (6).

Our remeshing solution runs anywhere between a few minutes to generate meshes containing thousands of vertices and a few hours for meshes containing hundreds of thousands of vertices. As the specific remeshing method is not the main focus of this work, and in fact, other surface remeshers could be adapted to our purposes, we have made no real effort to optimize the software.

### 3.2 Adaptive Remeshing

The disadvantage of meshing a geometric shape solely with unit edge length is that the smallest feature dictates the edge length for the entire mesh. This may increase the number of vertices to unrea-

sonable numbers, especially for meshes with fine detail. A possible solution would be to deviate from the pure concept of a connectivity shape and attach sparse information about feature size to the connectivity. Instead of specifying this for every edge, we would provide explicit edge length information only at the finest and the coarsest regions of the mesh. Locally, edges would still have equal lengths, but smoothly vary across the mesh, interpolating the explicit information. The challenge remains to (re)mesh in a manner such that the resulting edge lengths are close to those we would get from interpolating the sparse data on the connectivity graph.

### 3.3 Limitations

While Figure 6 demonstrates that it is possible to approximate given meshes by connectivity shapes surprisingly well – remember that these shapes contain no explicit geometric information whatsoever – it also exposes the limitations of the method: Without supplementary information, it will not be possible to generate a connectivity whose shape is *identical* to a target shape. This is especially noticeable in the teapot and the fine features of the dinosaur and horse. The limitations follow both from the definition of a connectivity shape, which is not unique, and the unit edge length objective of the remeshing method. The first might invert convexities and concavities and the second makes fine details expensive to capture.

## 4 HIERARCHICAL EMBEDDING

The iterative solver generating the connectivity shape, as described in Section 2.1, started with an initial embedding on the sphere, which is typically quite distant from the final solution. This results in a very slow procedure to generate the connectivity shapes of large meshes. To improve this, we adopt a hierarchical approach. In this setting the initial geometry for the solver at some level is taken as the result from a coarser level, which is a much better approximation of the final shape than a spherical embedding. Hence only the coarsest level uses a spherical embedding as the initial geometry.

The hierarchical method proceeds as follows: Starting from the given connectivity, we build from the connectivity *alone* a hierarchy of coarser and coarser connectivities. For each coarse level we compute new target edge lengths, not necessarily unit, which are derived from the finer level. The finest level, of course, has unit target edge lengths. When solving, we start with the coarsest level and embed its connectivity on the sphere. To accommodate non-unit edge lengths $l_{i,j}$ the spring energy term of (7) is refined to

$$E_S''\left(x \in \mathbb{R}^{n\times 3}\right) = \sum_{(i,j)\in\mathcal{E}} \left(||x_i - x_j||^2 - l_{i,j}^2\right)^2 \qquad (8)$$

After convergence at a coarse level, the initial geometry of the next (finer) level is generated from the coarse geometry. At each level we do not iterate through all the smoothing parameter values, but distribute the smoothing parameter interval over the levels. For example, for the hierarchical embedding of the horse as shown in the accompanying video with an optimal $\lambda$ of $0.0002$ we varied $\lambda$ on the coarsest level between $0.5$ and $0.05$, in the middle level between $0.02$ and $0.001$ and on the finest level between $0.0005$ and $0.0002$.

Since the hierarchy must be built from the connectivity alone, we chose to build the hierarchy levels by partitioning the mesh faces into patches as illustrated in Figure 7. This was done with the general purpose MeTis graph partitioning library [14]. MeTis partitions a connected graph into equal sized connected subgraphs. To group the *faces* and not the vertices into patches we feed the partitioning algorithm with the dual of the connectivity graph. This results in *vertex* separators, as opposed to the standard *edge* separators. MeTis minimizes the number of vertices on the patch boundaries, which results in nicely formed (small diameter) patches. From this we construct a coarser polygonal connectivity. The vertices of the new connectivity are the vertices of the original connectivity where more than two patches meet (large vertices in Figure 7(a)). In order to minimize the number of vertices in the coarse level, we merge vertices of degree three of the coarse level, which are adjacent also in the fine level, by local patch growing operations. The faces of the new connectivity are the patches, as shown in Figure 7 (b).

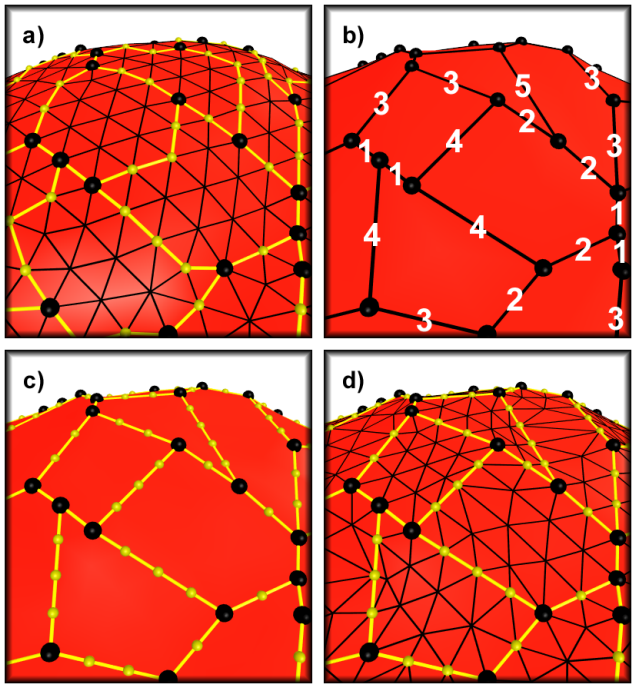


Figure 7: Stages of the hierarchical solver: (a) Partitioning of fine connectivity. Large vertices will be inherited by the coarse level. (b) Coarse level connectivity and target edge lengths. (c) Mapping back the solved coarse geometry on the fine connectivity. (d) The initial locations of the patch interior vertices

The target edge lengths of the finest level are unit. The target length of an edge in the next coarser level is taken to be the length of the corresponding edge path in the finer connectivity. In Figure 7(b) the edges are labeled with their target lengths calculated from the finer level.

Finally, we have to specify how to map the geometry of a coarse level back to the next finer level. Figure 7(c) illustrates this process. The locations of the vertices in the fine connectivity (fine vertices), which are preserved on the coarse level, are kept. Fine vertices on a coarse edge are distributed along the geometry of the coarse edge according to the target edge lengths on the finer level. All the remaining fine vertices are mapped into the interior of the coarse faces, corresponding to the patches they reside in. To find good initial locations for the latter fine vertices, we fix the fine vertices on coarse vertices or edges and perform a few "shape-preserving" iterations [5] based on the target edge lengths. The resulting locations are shown in Figure 7(d).

We varied the patch size for building the hierarchy between $3$ and $10$. It turned out that the hierarchical solver converged faster with small patch size. A patch size of $3$ outperformed the patch size of $10$ by about twenty percent. We compared the hierarchical embedding times of various meshes with that of the standard solver. Small meshes like the cow, giraffe and triceratops, required only two coarser levels and the hierarchical solver was four times faster. The cow's connectivity shape was computed hierarchically in $25$ seconds and the other two in $50$ seconds each on a Pentium III 600 MHz. For the dino and the horse meshes, the speedups with five

hierarchy levels were eight and eleven, respectively, resulting in 13 minutes for the dino and 40 minutes for the horse. Fifty percent of the embedding time for the horse was consumed on the finest level.

# 5 DISCUSSION

This paper has capitalized on the fact that a natural geometry may be associated with a given connectivity. This connectivity shape is defined as the geometry that minimizes some natural cost function of both the connectivity and the geometry. While showing how to work with these concepts in practice, there are still two open questions about the uniqueness of connectivity shapes, which we elaborate on here: (a) Is the definition unique ? (b) Does the iterative solver always find the desired solution ?

The definition of a connectivity shape is not unique. First of all the minimization problem (5) is, as expected, invariant under rigid body transformations and therefore adds six degrees of freedom to the solution. Furthermore, in the case of a non-triangular connectivity or a connectivity with border edges, the number of equations in (1) is less than the number of unknowns. Only in case of a triangular connectivity with genus zero does the number of equations equal the $3n - 6$ degrees of freedom.

The quadratic nature of (2) causes the in/out cap protrusion problem, which is not limited to caps of the size of a vertex neighborhood. Entire legs of the animal meshes can invert into the body of the mesh. Our regularization of the solution avoids some of this, but even if (5) has a unique minimum, it could be by accident a shape with the legs inside the body, and the desired shape (with the legs outside the body) could be a local minimum with slightly more energy. A topic of future work is to find this desirable "almost-global" minimum that maximizes the volume of the shape. This would automatically force extremities to protrude from the shape and avoid most self intersections.

A possible solution to the cap protrusion problem would be to augment the connectivity shape concept with another bit per vertex, denoting whether the surface at that point is concave or convex (the so-called "bump-bit"). Although we have not yet investigated this enough, we have some experimental evidence that this could solve the uniqueness problem.

There is a connection between connectivity shapes and the branch of combinatorial geometry called "Rigidity Theory" ([4], Chap. 6). A classical result of Cauchy implies that every *triangulated convex* polyhedron is uniquely determined by its edge lengths. For many years it was believed that non-convex triangulated polyhedra, while not uniquely determined by their edge lengths, are "rigid", namely, that there is no continuous transformation between the solutions which preserves the prescribed edge lengths. However, Connelly [3] and others showed that there are some pathological constructions where this does not hold (the so-called "flexible" polyhedra). In these few cases, however, the volume of the solution is fixed during the transition between solutions. Given a set of edge lengths of a graph, finding an embedding in $R^3$ which satisfies the edge length constraints, if one exists, is known to be NP-complete [17]. Determining whether there exists a *unique* solution may be done in polynomial time [11].

Since we incorporate a roughness term into our cost function, hence do not force precise edge lengths, none of this theory is really applicable. While this roughness term eliminates some of the solutions that minimize only the edge length component, it is still not clear whether the global minimizer of the complete cost function is unique.

The second aspect of uniqueness is whether the solver finds the desired solution, and does not get stuck in a local minimum far away from the global one. We normally achieve this by a good initial guess and cooling of the smoothing parameter. The spherical embedding as an initial guess has the advantage that we do not have any self intersections nor surface inversions to start off with. As the edge lengths are far from uniform in this initial mesh, we start with heavy smoothing to avoid minima with inverted surfaces.

For connectivities of genus greater than zero, the spherical embedding as initial geometry is suboptimal, since it will contain self-intersections. For the teapot shape of Figure 6 the spherical embedding worked well, but in general a more adapted approach is needed. Currently, we investigate how to map a connectivity graph with higher genus to a standard shape of the same genus.

An interesting connection between mesh connectivity and geometry may be made through the mesh Laplacian operator, as defined in (3). It has been observed in the past [9] that the $d$ eigenvectors of the corresponding Laplacian matrix that have the smallest non-zero eigenvalues may be used as coordinate vectors, forming a natural embedding of the connectivity graph in $R^d$. Hence a natural "spectral" embedding in 3D would be obtained using three such eigenvectors. It is easy to prove that this embedding minimizes the total squared Euclidean edge length in $R^3$, subject to the constraint that the norm of the three coordinate vectors is unit. Our experiments with this "spectral" embedding have resulted in pleasing 2D embeddings for meshes of disk topology (using two eigenvectors), but often distorted 3D embeddings for meshes of sphere topology (using three eigenvectors). This is probably because our connectivity graphs just describe a two dimensional surface in a three dimensional space. Interestingly enough, the eigenvectors of the Laplacian matrix have also been used for mesh geometry coding [13].

# 6 OTHER APPLICATIONS

Connectivity shapes proved very useful for the visualization of connectivity graphs describing a two manifold. In this section we mention some other potential applications of connectivity shapes.

## 6.1 Shape Modeling

The notion of a connectivity shape suggests using it as a modeling tool. Envision an interactive modeling system, in which the modeling primitives are connectivity operations only, e.g. spray on some vertices, change the connectivity by edge flips, collapse edges. It remains to be seen whether such a tool would be intuitive to use with enough expressive power to generate a rich variety of shapes.

## 6.2 Connectivity Creatures

It is entertaining to generate the connectivity shapes of existing mesh connectivities, as we did in Figures 1(c), 4(a,b), and 8 and see how they turn out. These *connectivity creatures* will not have the shape of the meshes whose connectivity was used, but some distortion of it, usually with exaggerated features as commonly seen in caricatures. Connectivity creatures certainly have some entertainment value and there exists a large volume of 3D content that can be manipulated this way without any extra modeling.

## 6.3 Mesh Coding

Another possible application for connectivity shapes is 3D mesh coding. For storage and transmission purposes it is important to represent a mesh as compactly as possible. This problem has received a lot of attention recently, and much of the effort has focused on coding the connectivity component efficiently [18, 19, 8, 16, 15, 12]. While this has resulted in codes requiring less than 4 bits per vertex (bpv) on the average, the geometric information now dominates the coding cost, and, assuming 10 bpv pre-quantization, even efficient predictive coding [18, 19] is not able to reduce these 30 bpv to much less than 15 bpv on the average. It would be very beneficial to reduce the geometric component of a mesh further. A possible way to achieve this are connectivity shapes.

Using connectivity shapes (with or without "bump bits") for coding means that the code will be lossy, in the sense that the recovered

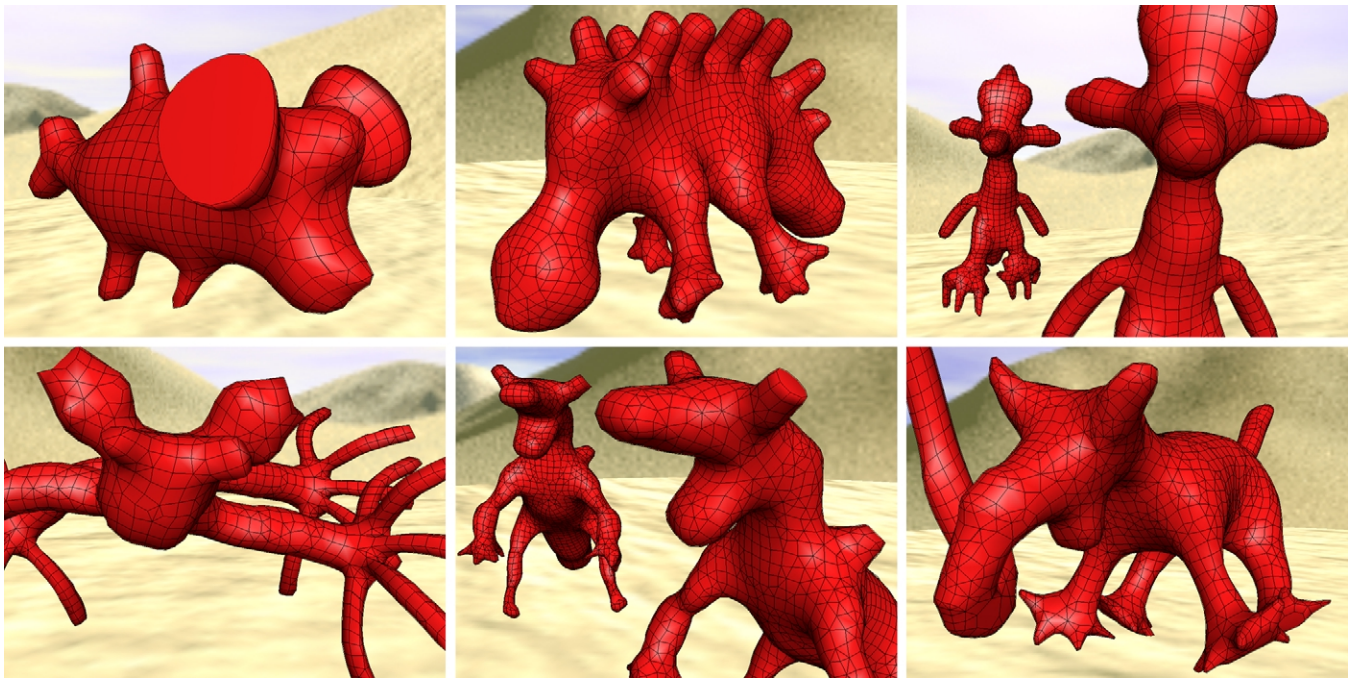

Figure 8: Connectivity creatures: tuna fish, stegoratops, penguin, gila monster, dragon, and elephant.

(decoded) mesh will not be identical to the original, both in connectivity and geometry. While loss is undesirable for some applications, and the first mesh coding algorithms made an effort to avoid this, it is becoming more and more acceptable to code meshes in a lossy manner, as long as the decoded version is sufficiently close to the original. In the context of connectivity shapes, this means that the coding algorithm is the remeshing procedure, the code is some efficient coding of *only* the connectivity information, and the decoding algorithm is the connectivity shape generation procedure. A remeshed shape often has a much larger number of vertices than the original mesh. While this increases the amount of connectivity information that needs to be encoded, preliminary experiments suggest that overall this method is still cost effective. This is especially true since connectivity coding algorithms favor highly regular meshes like those produced by our remeshing procedure. An approach using adaptive remeshing, as envisioned in Section 3.2, which reduces the number of vertices, but retains sparse geometric information, would combine the best of both worlds.

## Acknowledgments

Thanks to Greg Turk for supplying us with his retiling software and to Zachi Karni for technical support. This work was funded by the European MINGLE project HPRN-CT-1999-00117.

## References

[1] N. Amenta, M. Bern, and M. Kamvysselis. A new Voronoi-based surface reconstruction algorithm. In *SIGGRAPH'98 Conference Proceedings*, pages 415–421, 1998.

[2] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In *Computing in Euclidean Geometry*, pages 71–78. World Scientific, 1995.

[3] R. Connelly. A counterexample to the rigidity conjecture for polyhedra. *Inst. des Hautes Études Scientifiques Publications Mathematiques*, (47):333–338, 1977.

[4] P. R. Cromwell. *Polyhedra*. Cambridge University Press, 1997.

[5] M. S. Floater. Parameterization and smooth approximation of surface triangulation. *Computer Aided Geometric Design*, 14:231–250, 1997.

[6] P. J. Frey. About surface remeshing. In *Proceedings of the 9th International Meshing Roundtable*, pages 123–136, 2000.

[7] P. Gajer, M. T. Goodrich, and S. G. Kobourov. A fast multi-dimensional algorithm for drawing large graphs. In *Graph Drawing'00 Conference Proceedings*, pages 211–221, 2000.

[8] S. Gumhold and W. Strasser. Real time compression of triangle mesh connectivity. In *SIGGRAPH'98 Conference Proceedings*, pages 133–140, 1998.

[9] K. M. Hall. An r-dimensional quadratic placement algorithm. *Management Science*, (17):219–229, 1970.

[10] D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. In *Graph Drawing'00 Conference Proceedings*, pages 183–196, 2000.

[11] B. Hendrickson. Conditions for unique graph realizations. *SIAM Journal of Computing*, (21):65–84, 1992.

[12] M. Isenburg and J. Snoeyink. Face Fixer: Compressing polygon meshes with properties. In *SIGGRAPH'00 Conference Proceedings*, pages 263–270, 2000.

[13] Z. Karni and C. Gotsman. Spectral compression of mesh geometry. In *SIGGRAPH'00 Conference Proceedings*, pages 279–286, 2000.

[14] G. Karypis and V. Kumar. METIS - a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. *Version 4, University of Minnesota.*, Available on WWW at URL http://www-users.cs.umn.edu/ karypis/metis/.

[15] B. Kronrod and C. Gotsman. Efficient coding of non-triangular meshes. In *Proceedings of Pacific Graphics*, pages 235–242, 2000.

[16] J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1), 1999.

[17] J. B. Saxe. Embedding of weighted graphs in k-space is strongly NP-hard. In *Proc. 17th Allert. Conf. in Commun. Control and Comput.*, pages 480–489, 1979.

[18] G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, 1998.

[19] C. Touma and C. Gotsman. Triangle mesh compression. In *Graphics Interface'98 Conference Proceedings*, pages 26–34, 1998.

[20] G. Turk. Re-tiling polygonal surfaces. In *SIGGRAPH'92 Conference Proceedings*, pages 55–64, 1992.

[21] W. T. Tutte. How to draw a graph. *Proceedings of London Mathematical Society*, 13(743–768), 1963.

[22] C. Walshaw. A multilevel algorithm for force-directed graph drawing. In *Graph Drawing'00 Conference Proceedings*, pages 171–182, 2000.