

# Interactive Planarization and Optimization of 3D Meshes

Roi Poranne<sup>1</sup>

Elena Ovreiu<sup>2</sup>

Craig Gotsman<sup>1</sup>

<sup>1</sup> Computer Science Department, Technion, Haifa, Israel

<sup>2</sup> Université de Lyon, France

---

## Abstract

*Constraining 3D meshes to restricted classes is necessary in architectural and industrial design, but it can be very challenging to manipulate meshes while staying within these classes. Specifically, polyhedral meshes - those having planar faces - are very important, but also notoriously difficult to generate and manipulate efficiently.*

*We describe an interactive method for computing, optimizing and editing polyhedral meshes. Efficiency is achieved thanks to a numerical procedure combining an alternating least-squares approach with the penalty method. This approach is generalized to manipulate other subsets of polyhedral meshes, as defined by a variety of other constraints.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling.

**Keywords:** planarization, polyhedral meshes, shape optimization.

---

## 1. Introduction

Working with flat materials such as glass or plywood is a typical scenario in architectural and industrial design. Such materials impose serious constraints on the creative freedom of a designer, as the space of these constructions is the space of *polyhedral meshes*, namely, meshes with planar faces, which is known to be very limiting. Beyond it being difficult to generate such meshes, it is even more difficult to manipulate them while staying within the class.

Contemporary modeling methods for polyhedral meshes usually start with a freeform surface. Once the designer is satisfied with the shape, an elaborate meshing scheme is employed to convert the freeform into a mesh with planar faces.

In this paper, we provide an *efficient* solution to the following problem: given a general (non-polyhedral) 3D mesh, find the closest polyhedral mesh (with planar faces) that has the same or similar topology. A solution to this problem enables an interactive system in which the designer has a lot of freedom.

In addition, we briefly discuss one special case of the main problem of this paper. This is the design of a polyhedral surface such that its projection on a plane, sometimes called its *picture*, is fixed. We also show that our approach can be generalized to accommodate other constraints, and exemplify it with a new “as-similar-as-possible” planarization procedure.

## 2. Related Work

### 2.1 Mesh Optimization

The issue of the planarity of 3D mesh faces has been treated before. In most cases, the goal was to mesh, or remesh, a given freeform surface so that the resulting mesh is polyhedral. Cohen-Steiner *et al.* [CSAD04] describe a method in which the surface is partitioned into a user-defined number of almost flat regions. A plane, called the *shape proxy* of the region, is fit to each region and then used as a basis for a new, simplified mesh with *almost* planar faces. Later, Cutler and Whiting [CW07] added an iterative process to guarantee that the resulting faces would be planar.

In both of these systems, the user can control the number of faces and their density in the result but cannot dictate the mesh connectivity (its edge structure). While this is not necessarily a drawback, in some cases a regular mesh is desirable. Liu *et al.* [LPH\*06] and Wang *et al.* [WLY\*08] show how a surface may be meshed into a planar quad-dominant (PQ) mesh and a planar hexagonal (P-Hex) mesh, respectively. The two algorithms are quite similar: An almost polyhedral mesh is first generated from the surface based on differential geometric entities (PQ meshes are based on conjugate networks, and P-Hex meshes on the Dupin indicatrix). A second step involves non-linear optimization, where the vertices of the mesh are repositioned to make the faces planar. The latter step seems to dominate the runtime, and does not scale well with mesh size. We will describe an alternative optimization method which generates a solution in essentially linear time.

Alexa and Wardetzky [AW11] show how to construct a Laplacian operator on non-triangular meshes. Their operator can be used, as in the triangular case, for smoothing a mesh by so-called Laplacian smoothing, which is equivalent to mean-curvature flow. Such a flow gradually decreases the mean curvature at all points of the mesh, following the gradient of the mesh surface area, eventually leading to a mesh with minimal surface area. As a side effect of their construction, Alexa and Wardetzky were able to devise a related operator that measures the planarity of faces. With this new operator, they obtain a *planarizing flow*, that is, a geometric flow that flattens faces. This flow, however, still boils down to iteration of a slow non-linear optimization problem.

Another approach that can be applied only to quad meshes is described by Hoffmann [Hof11], who shows that a single quad can be deformed without changing anything but its neighbors and without violating the planarity constraints. Unfortunately, this does not generalize to a method to deform a complete quad mesh.

In parallel to our work, Bouaziz *et al.* [BDS\*12] recently presented a general framework for *projecting* meshes on a set of constraints, e.g. having planar faces. Their framework is similar to the one presented in Section 6.2 of this paper, and their solver is also similar to ours.

## 2.2 Interactivity

All the methods mentioned in the previous section cannot be used in an interactive, edit-and-observe modeling scenario, due to their lengthy runtimes. A different approach that allows interactive editing was presented by Yang et al. [YYPM11], who treat the space of polyhedral meshes with given topology as a manifold embedded in a high dimensional Euclidean space. Given a polyhedral mesh - a point on this manifold - we wish to *explore* its neighborhood, hence the term “shape space exploration”. Naturally, exact computation of the neighborhood will be difficult, so an osculant to the manifold at the point, which approximates the neighborhood and is simpler to traverse, is constructed instead. Sadly, this approximation is too crude to allow straying too far from the original polyhedral mesh without violating the planarity constraints. To rectify this, successive projections to the manifold must be made. The authors suggest using the same method as Liu *et al.* [LPH\*06] which, again, is very time-consuming.

## 3. Contributions

Our main contributions are:

1. An extremely simple, essentially linear-time optimization algorithm to generate a polyhedral mesh based on a given control mesh.
2. A novel view of *graph liftings* for architectural design.
3. A simple approach to accommodate a variety of other constraints.

## 4. Mesh Planarization

### 4.1 Problem Statement

Let  $P = \{p_i\}_{i=1}^{N_V}$  be the geometry of a set of vertices of a given *control mesh* and  $Q = \{q_i\}_{i=1}^{N_V}$  the geometry of the same set of vertices of a *solution mesh* (to be found). Let  $F = \{f_j\}_{j=1}^{N_F}$  be the set of faces of the control mesh, where each face is given as an ordered (oriented) list of vertices. Since the topology of the control and solution meshes is identical, so are their face sets. We write  $p_i \in f_j$  if the vertex  $p_i$  is incident to  $f_j$ . Abusing notation slightly, we will also write  $f_j \in p_i$  to state the same in reverse. In addition, we define the set of *corners*  $\mathcal{C}$  of the mesh as the set of indices of incident vertex-face pairs in the mesh. In other words,  $(i, j) \in \mathcal{C}$  if  $p_i \in f_j$  (or  $f_j \in p_i$ ).

A very general way of describing the optimization problem is as follows:

$$\begin{aligned} \min_Q \quad & \text{dist}(P, Q) + \lambda E_{reg}(Q) \\ \text{s.t.} \quad & \forall j \{q \mid q \in f_j\} \text{ are co-planar} \end{aligned} \quad (1)$$

where  $\text{dist}$  is some distance function between the mesh surfaces defined by  $P$  and  $Q$ , and  $E_{reg}$  is a regularization term defined on the result. For the sake of clarity, we will limit our discussion to a point-to-point Euclidean distance, and drop the regularization term for now.

The planarity constraint on each face can be expressed in terms of the vertex geometry of the face alone. For example, the volume of a tetrahedron defined by four vertices of a quad is zero iff the vertices are coplanar, leading to a

trilinear constraint on the coordinates of the four vertices. In the general case of a face containing  $v > 4$  vertices, coplanarity of the face vertices is obtained iff

$$\det(Q^T Q) = 0 \quad (2)$$

where  $Q$  is the  $v \times 4$  matrix of the vertex geometry in homogeneous coordinates. This is a polynomial constraint of order six, which is more difficult to work with.

A key observation is that by adding the normals to the planes of the faces as variables, the optimization problem can be cast as a least squares problem with only bilinear and norm constraints. Specifically, we characterize each face  $f_j$  by its *unit* normal vector  $n_j$  and the distance  $d_j$  of its plane from the origin, forming the sets  $N$  and  $D$  of normals and distances respectively. We require that for each face  $f_j$  and for each vertex  $q_i \in f_j$ ,  $q_i$  will lie on the plane defined by  $(n_j, d_j)$ . Problem (1) (without the regularization term) may then be stated as

$$\begin{aligned} \min_{\substack{q_i \in Q \\ n_j \in N \\ d_j \in D}} \quad & \sum_{i=1}^{N_V} \|p_i - q_i\|^2 \\ \text{s.t.} \quad & n_j \cdot q_i + d_j = 0, \quad \forall (i, j) \in \mathcal{C} \\ & \|n_j\|^2 = 1 \end{aligned} \quad (3)$$

We will refer to these constraints simply as *distance from plane* constraints (DFP). We mention here that Deng et al. [DPW11] also proposed a similar formulation of the problem for designing planar webs. We improve on this by showing, in the next section, how to take advantage of the form (3) of the problem to devise a very efficient solver.

### 4.2 An Alternating Solver

While problem (3) may be solved using a generic solver, it does not scale well, as do other non-linear problems. Most of the commercial solvers we tried, when confronted with this problem, took a long time to handle a mesh of more than few hundred vertices. Implementing a dedicated solver requires considerable expertise. We now show how to circumvent this complexity, using only the simplest tools of linear algebra.

First we apply the quadratic penalty method [NW99, Ch. 17.1], rewriting problem (3) as

$$\begin{aligned} \min_{\substack{q_i \in Q \\ n_j \in N \\ d_j \in D}} \quad & \mu \sum_{i=1}^{N_V} \|p_i - q_i\|^2 + (1 - \mu) \sum_{(i,j) \in \mathcal{C}} (n_j \cdot q_i + d_j)^2 \\ \text{s.t.} \quad & \|n_j\|^2 = 1 \end{aligned} \quad (4)$$

where  $\mu$  is the penalty coefficient. Starting with  $\mu = 1$  and an initial guess, the method requires successively solving (4), where in each iteration  $\mu$  is gradually reduced, and the previous solution is used as a new initial guess. This iteration is guaranteed to converge to a local minimum of (3) under very mild conditions.

Problem (4) still cannot be solved by simple means. But consider the following: if the  $n_j$ 's are fixed, then (4) becomes quadratic and may be solved by a (sparse) *global* linear system.

Although the global linear system can be solved very quickly, we can further simplify (4): if the  $n_j$ 's and the  $d_j$ 's are fixed, then the problem becomes completely separable in the remaining variables. That is, we can find each  $q_i$  by solving a small *local* problem with  $q_i$  as the only variable. Namely,

$$\min_{q_i} \left[ \mu \|p_i - q_i\|^2 + (1 - \mu) \sum_{(i,j) \in C} (n_j \cdot q_i + d_j)^2 \right] \quad (5)$$

The same happens if we fix the  $q_i$ 's. In that case finding each pair  $(n_j, d_j)$  amounts to solving the *local* problem

$$\begin{aligned} \min_{n_j, d_j} \quad & \sum_{(i,j) \in C} (n_j \cdot q_i + d_j)^2 \\ \text{s.t.} \quad & \|n_j\|^2 = 1 \end{aligned} \quad (6)$$

Problem (5) is a simple least squares problem in  $q_i$  and can be solved using the standard pseudo-inverse technique. It is exactly the problem of finding a point  $q_i$  which is as close as possible (in the Euclidean norm) to a set of given planes. Similarly, problem (6) is exactly that of fitting a plane  $(n_j, d_j)$  to a set of points, and can be solved using eigen-decomposition. The details can be found in Algorithm **Planarize**.

Thus we alternate between fixing one set of variables and solving for the other set repeatedly, until convergence. Following [SA07] and [LZX\*08], we call the first approach, involving a global linear system for  $(q_i, d_j)$  and a local system for  $n_j$  a ‘‘local/global’’ (L/G) scheme. The second approach, involving a local system for  $q_i$  and a local system for  $(n_j, d_j)$  is called ‘‘local/local’’ (L/L). We will refer to both of them as the local schemes. The solution to (4) using a generic non-linear solver will be called simply the ‘‘global’’ solution.

Since both the local schemes are not strictly of the alternating least squares type, we cannot prove that they will achieve a local minimum of (4), (although it is easy to see that they do converge). While this is necessary for the penalty method to work, our experiments show that in well-behaved cases, both solvers achieve the same minima. Furthermore, even in difficult situations, the final energy is always zero, meaning that the planarity constraints are satisfied.

**Regularization.** Starting from the general problem in (1), many additional regularization terms can be added. However, to benefit from the local approaches, the term must have a local form which can be incorporated to each problem in the alternating steps. The most obvious regularization term that comes to mind is Laplacian smoothing, where the distance between a vertex and the average of its neighbors is minimized. As a global *energy*, it is expressed by

#### Algorithm Planarize (local/local)

Compute  $Q (P, N, D, F, \mu_0, r, k)$

$Q = P$

$\mu = \mu_0$

**do**  $k$  times

**for**  $j = 1$  to  $N_f$

$\bar{Q} =$  Rows of  $Q$  corresponding to vertices of  $f_j$

$M =$  Average of columns of  $Q$

$\bar{Q}_c = Q - M$

$N_j =$  Eigenvector with smallest eigenvalue of  $\bar{Q}_c^T \bar{Q}_c$

$d_j = -M^T N_j$

**end**

**for**  $i = 1$  to  $N_v$

$\bar{N} =$  Rows of  $N$  corresponding to faces of  $q_i$

$\bar{D} =$  Entries of  $D$  corresponding to faces of  $q_i$

$A = \mu I + (1 - \mu) \bar{N}^T \bar{N}$

$b = \mu p_i - (1 - \mu) \bar{D}^T \bar{N}$

Solve  $A q_i = b$  for  $q_i$

**end**

$\mu = r\mu$

**End**

$$E^{lap,q}(q_1, \dots, q_n) = \sum_{i=1}^n \left( q_i - \frac{1}{a} \sum_{k \in \text{neigh}(q_i)} w_{i,k} q_k \right)^2 \quad (7)$$

where  $\text{neigh}(q)$  is a list of indices of *vertex*-neighbors of  $q$  and  $a = \sum_{k \in \text{neigh}(q_i)} w_{i,k}$ . This energy can be separated, as required by our method, into an energy term for each vertex

$$E_i^{lap,q}(q_i) = \left( q_i - \frac{1}{a} \sum_{k \in \text{neigh}(q_i)} w_{i,k} q_k \right)^2 \quad (8)$$

A similar energy is occasionally used for the normals of the faces. That is, the normals of faces are smoothed as well. Locally, this can be written as

$$E_j^{lap,n}(n_j) = \left( n_j - \frac{1}{b} \sum_{k \in \text{neigh}(f_j)} w_{j,k} n_k \right)^2$$

where  $\text{neigh}(f)$  is a list of indices of *faces*-neighbors of  $f$  and  $b = \sum_{k \in N_{f_j}} w_{j,k}$ . Both of these energies are quadratic, and adding them to the objective functions in (7) and (8) does not make the algorithm any more complicated.

Several other similar regularization terms have been proposed in the literature, such as minimizing the change in the Laplacian [ZSW11]. We will discuss more complex terms in Section 6, once we establish a generalization of the alternating schemes.

## 5. Evaluation

**Implementation Details.** The global solution to (3) was computed using the KNITRO [BNW06] general-purpose

solver, which uses Sequential Quadratic Programming (SQP). This is the fastest generic solver we could find (also supported by the Benchmarks for Optimization Software: <http://plato.asu.edu/bench.html>). We provided analytic gradients, but the Hessians were computed numerically, based on their sparsity pattern, which is faster than evaluating them analytically. Initial planes were found using the second loop of Algorithm **Planarize**.

The local schemes were implemented in C++. We used Eigen v3 [GJ10] for matrix computation and [OMP] for parallelizing the two loops in a multi-core environment. Since both loops do not require any synchronization, using OpenMP to parallelize them is extremely simple: we had only to add the line.

```
#pragma omp parallel for schedule (dynamic,1)
```

before each loop. The resulting speedup was 3-5 $\times$  on a 4-core Intel i7 CPU with 4GB RAM. There is still a lot more room for improvement, since we did not optimize calls to Eigen, and have observed many redundant memory allocations.

**Planarity measure.** All of the constraints mentioned in Section 4.1 force planarity in different ways such that a planar face will always measure exactly zero, but the behavior of the different measures on non-planar faces is considerably different. For a fair comparison between the methods, we always measure non-planarity of a computed face as the smallest eigenvalue obtained by principle component analysis (PCA) of the face vertex geometry.

**Comparison to other planarity constraints.** As mentioned in Section 4.1, there are a number of ways to measure the planarity of faces. In addition, Liu *et al.* [LPH\*06] measure the planarity of a quad by the sum of its angles, which is  $2\pi$  iff the quad is planar. Wang *et al.* [WLY\*08] measure the planarity of a hexagon by measuring the planarity of the quads defined by each four consecutive vertices, which, in turn, is measured by the volume of the tetrahedron spanned by those four vertices. These two methods are easily generalized to any sized polygon.

These planarity constraints, as well as the one described in (2), and the DFP constraints we use in (3), are equivalent in the sense that they must define the same feasible region in space, thus the same global minimum. However, they differ in practice when an iterative solver is used. Each set of constraints define a different gradient descent path which may or may not end up at a feasible point and the number of steps needed to reach a feasible point varies. Fig. 7 and 8 demonstrates the convergence of a generic optimizer using the various sets of constraints with a point-to-point distance as the objective. Using DFP constraints, the optimizer always reaches a feasible point, and in the least number of steps. Using other constraints causes the optimizer to get stuck, in many cases, away from the feasible domain. Thus we conclude that even if a generic global solver is to be used, it is best to apply it to the problem as described using the DFP constraints (3).

The other aspect of the optimization that should be evaluated is the final energy that was achieved. This is meaningful only if two solutions have the same measure of planarity. Therefore, the meshes we show in Fig. 7 and 8 are not the final results of the optimization, but intermediate

results acquired just after the mesh achieves a given level of planarity (as measured using PCA).

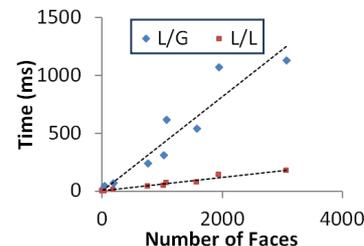
**Global vs. L/G vs. L/L.** Using DFP constraints improves the convergence properties and the results of the optimization, but the major benefit in using them is the option of using the local schemes. As shown earlier, these approaches are very simple to implement, and their runtimes are several orders of magnitudes smaller than the alternatives. For example, it took 5 and 12 seconds to optimize Torus1 and Torus2 from Fig. 7 using a global optimizer and only 70 and 144 milliseconds to optimize using the L/L scheme. Hence, the only concern regarding whether to use them or not is related to the final result. Their behavior, or more precisely, the behavior of the penalty method, is directly related to the rate of change of  $\mu$ . It has to be reduced “slowly enough” for the local schemes to converge to the same minimum as the global solver. In our tests, “slowly enough” was determined experimentally, although a more deterministic method exists (see [NW99, Ch. 17.1]). Our experiments show that setting  $r = 0.9$  in the L/L case and  $r = 0.7$  in the L/G produces good results after 100 and 30 iterations respectively.

The runtime of Algorithm **Planarize** depends mostly on  $N_V, N_F$  and the number of iterations  $k$ , and the dependence is linear. We have observed that for  $\mu < 10^{-4}$ , the constraints are satisfied with a tolerance less than  $10^{-6}$ , which is the default tolerance in most commercial optimizers. Therefore,  $k$  depends only on  $\mu_0$  and  $r$ . We have also observed that the results are not improved by setting both  $t_0$  and  $r$  to a value larger than 0.9. Hence  $t_0$  and  $r$  can be fixed. We therefore conclude that the algorithm essentially depends linearly on  $N_V + N_F$ . The times measured to run 100 L/L iterations on the meshes of Fig. 7 and 8 are graphed in Fig. 2, where the linear behavior is evident.

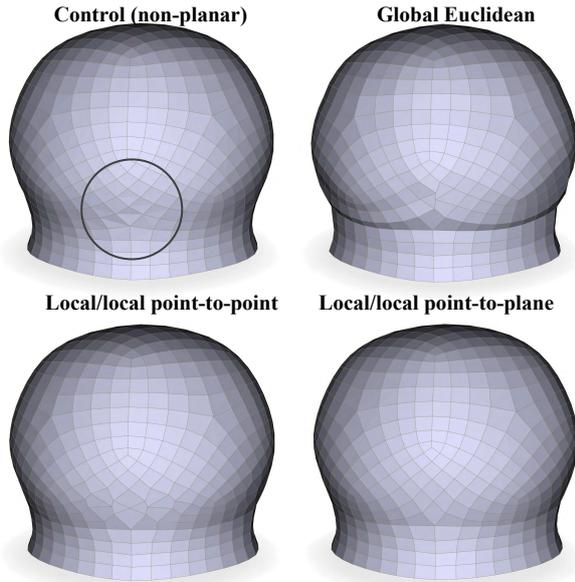
Analysis of the time complexity for the L/G case is similar, only in this case a sparse linear system must be solved at each iteration. While theoretically this requires quadratic time, in practice we see from Table 1 and Fig. 2 the time is linear in the number of faces. This is perhaps due to the specific sparsity pattern of this particular linear system.

Model (fig)	Faces	Vertices	L/G Time	L/L Time
Level 1 (2a)	12	20	16	1
Level 2 (2b)	48	64	44	4
Level 3 (2c)	192	224	64	16
Level 4 (2d)	768	832	238	42
Quadmesh 1	1027	1154	310	50
Torus1	1080	2160	618	70
Quad mesh 2	1579	1791	536	76
Torus2	1944	3888	1068	144
Level5	3072	3200	1126	176

**Table 1:** Runtime (msec) of 30 L/G iterations and 100 L/L iterations



**Figure 1:** Plot of Table 1.



**Figure 2:** Decomposing the Euclidean norm into components may be preferable. Minimizing only the point-to-plane distance produces a solution closer to the control mesh (of 901 vertices and 880 quads). Note the circled region, where the less planar faces reside, and which causes most of artifacts in the planarized results. It took 38 msec to run 100 iterations with our L/L solver and 4 sec to solve using the global solver.

**Interactive Tweaking.** It has been noted before that it is sometime preferable to decompose the point-to-point Euclidean norm to a distance from the surface normal and from the tangent plane (also known as point-to-plane). If  $n_q$  is the unit normal at the vertex  $q$ , the (squared) distance of a point  $q'$  from the tangent plane at  $q$  is simply:

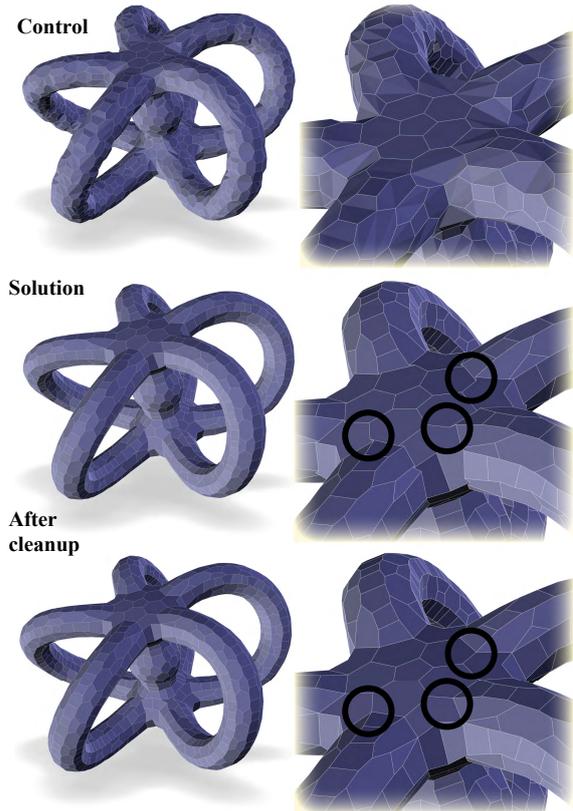
$$D_t(q, q') = ((q - q') \cdot n_q)^2$$

and the (squared) distance from the normal is:

$$D_n(q, q') = \|(q - q') \times n_q\|^2$$

which are still quadratic if either  $q$  or  $n$  are fixed. In our implementation, we replaced the Euclidean norm with a combination of  $D_t$  and  $D_n$ . An example is shown in Fig. 2 – both the global solution and the local solution with Euclidean distance generate artifacts. In contrast, penalizing only distance from the tangent plane produces nice results. As shown in the accompanying video, reaching the best solution is possible by interactive tweaking of the parameters of the problem.

**Edit-and-Observe.** One of the main difficulties of modeling with freeforms via control points is that it is hard to predict the effect of moving a control point, especially for a novice. This is the case whether using an interpolation scheme, or to a greater extent, using an approximation scheme. The designer must constantly reposition the vertices of the control mesh, change their weights, change their continuity and so on. This process is feasible only because of the immediate result provided. The same is true when working with polyhedral meshes; the planarity constraint



**Figure 3:** Using an interactive system, the designer can spot artifacts (circled), remove them, and immediately see the result.

causes them to be somewhat unpredictable. So a real-time response is critical, and this is feasible due to efficiency of the local schemes.

We expect a typical design session would start from a non-polyhedral mesh constructed in a generic modeling system. Once the designer is satisfied with the general shape of the model, the design mode will be changed to polyhedral mode, by meshing or otherwise. The model would then become a *control* (wireframe) mesh, and the resulting polyhedral *solution* would be seen simultaneously. Sometimes a few artifacts may occur in the result, which can be fixed manually by the user or automatically. Figure 3 shows a mesh before and after this cleanup process. The solution has a few short edges that seem redundant. Using an automatic edge collapse method, all short edges are removed, and the optimization process runs again. The result is a clean polyhedral mesh. If the result is still not satisfying, the control mesh can be completely *remeshed* using one of the methods mentioned in this paper, and then optimized.

In the accompanying video, we show a very simple mesh modeling program we have developed to deform a control mesh and display the polyhedral result. The video shows that using our local solver, real-time deformation of polyhedral meshes is indeed possible.

Currently, we can use our software to locally reposition vertices, and set the optimization parameters interactively. Using “soft selection” we can move a single vertex with a variable size of its neighborhood. If the original control mesh generated a good solution, then small deformations will typically also generate good solutions. Our intention is to integrate our solver into other deformation schemes,

such as skeletal or cage based deformations, and into meshing schemes such as those discussed in this paper.

## 6. Additional Results

### 6.1 Optimal Lifting

We now discuss a special case of the main problem presented in this paper which turns out to be related to the classic “lifting” problem of planar graphs. This could have architectural applications in its own right. A complete exposition on this problem and its solution can be found in [Ric96], and it can be stated as follows: Given a 3-connected planar graph  $G$  embedded in the plane, can we assign heights, or *lift* each vertex, such that each face remains planar? This is not always possible, in a non-trivial way, and a classical example is the so-called Schoenhart graph (see embedded image on right).



A solution to the lifting problem, if it exists, can be computed by following the same ideas as presented in Section 5: add the normals of the faces to the problem as variables and get (3) again, only with less variables. Since the  $x$  and  $y$  coordinates of each point are known, if we fix the  $z$ -coordinates of the normals (say to 1) and then safely ignore their unit length constraints, we would be left with nothing more than a linear least squares problem with linear constraints, whose *global* minimum may be obtained by solving a *linear* system. For the sake of completeness, we detail here the solution to this problem.

Denote by  $\hat{z}_i$  the  $z$ -coordinates of the vertices to be found,  $n_j^x$  and  $n_j^y$  the  $x$  and  $y$  components of the normal to face  $f_j$  (the  $z$  component is set to 1) and  $d_j$  as before. Define the variable vector  $w$  as a  $(N_v + 3N_f) \times 1$  column vector obtained by concatenating  $\hat{z}_i$ ,  $n_j^x$ ,  $n_j^y$  and  $d_j$ . The energy function is

$$\sum_{i=1}^{N_v} (z_i - \hat{z}_i)^2 = \|Aw - b\|^2,$$

$$A = \begin{pmatrix} I_{N_v} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}, \quad b = \begin{pmatrix} z_i \\ \mathbf{0} \end{pmatrix}$$

where  $I_{N_v}$  is the  $N_v \times N_v$  identity matrix, and  $\mathbf{0}$  is a vector or a matrix of appropriate size. For each corner  $(i, j)$  we have the linear constraint  $\hat{z}_i + x_i n_j^x + y_i n_j^y + d_j = 0$ . Let  $B$  be the  $c \times (N_v + 3N_f)$  matrix ( $c$  is the number of corners) and  $d$  the  $c \times 1$  vector such that  $Bw = 0$  expresses all of these equations. Then the optimization can be written as

$$\min_w \|Aw - b\|^2$$

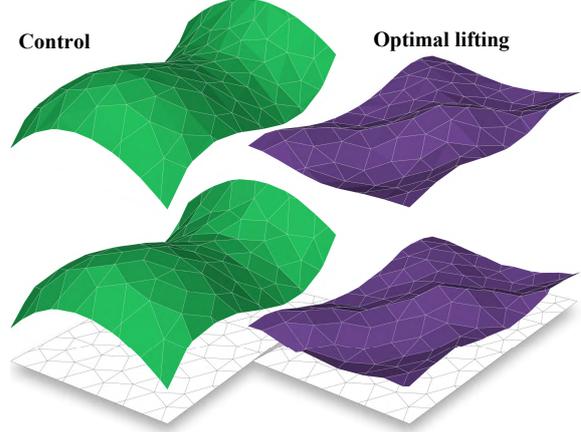
$$\text{s. t. } Bw = 0$$

The solution to this problem is known to be

$$\begin{pmatrix} w \\ \lambda \end{pmatrix} = \begin{pmatrix} A^T A & B^T \\ B & 0 \end{pmatrix}^{-1} \begin{pmatrix} A^T b \\ d \end{pmatrix}$$

where  $\lambda$  are Lagrange multipliers used to find the solution. Since we are ultimately interested only in  $\hat{z}$ , we can simply take the first  $N_v$  elements of the solution.

We provide a few examples of such liftings in Fig. 4. It is important to remember how limited liftings are when designing them. Even if the embedding is liftable, the



**Figure 4:** Optimal liftings approximating non-polyhedral control meshes with the same topology and  $(x,y)$  projection.

space of liftings can have very low dimension. For example, it is easy to show that the space of liftings of a 3-regular graph (having any planar geometry) is only 4-dimensional.

We can analyze this space of liftings from an algebraic point of view: it is represented by the nullspace of the constraint matrix  $B$ . Any basis of  $\text{null}(B)$  can be used to construct all the valid liftings. This suggests a *different* modeling scheme, where the designer is given a small set of “basis functions” from which she can create a lifting. However, not every spanning subset is geometrically meaningful. In what follows, we briefly describe three choices of such sets, where a function is associated with each vertex. Results are shown in Fig. 5.

**Fundamental Liftings.** We construct one function per vertex as follows. For each vertex  $p_i$  in the mesh, we define target heights  $z_j = \delta_{ij}$ . The function for  $p_i$  is then the optimal lifting for these target values of  $z_i$ , which we call the *fundamental lifting of vertex  $i$* .

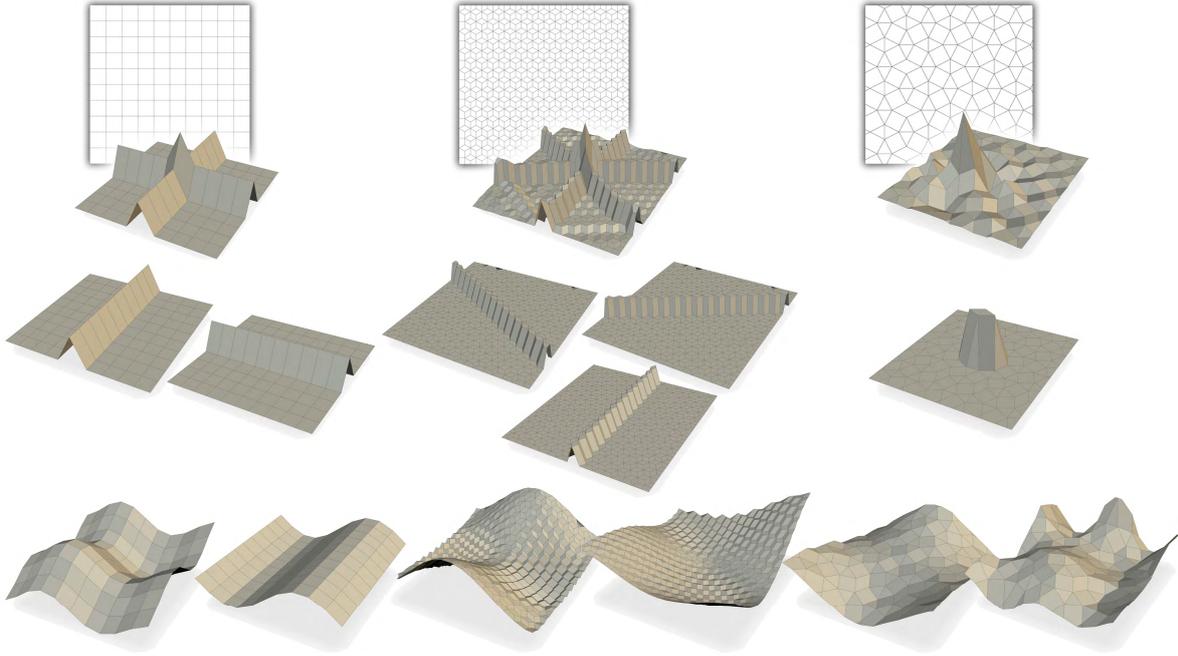
**Sparse Liftings.** While the aim of the fundamental lifting is to move one vertex while the rest stay in place as much as possible, we can try something a little different: Move several vertices together, while the rest do not move at all, forming *sparse liftings*. This corresponds to *sparse* vectors of  $\text{null}(B)$ . Generally, finding such vectors is an NP-hard problem, but the local nature of planar graphs allows us to find sparse liftings for small neighborhoods and then stitch them to form sparse liftings for the entire graph.

**EigenLiftings.** Both previous approaches give the designer a tool to fine-tune the lifting, using delta-like functions. To add globally smooth liftings to the set, we resort to the eigenvectors of the combinatorial graph Laplacian,  $L$ . Of course, the eigenvectors must be constrained to the space of liftings, and these can be found using the constrained Rayleigh quotient:

$$\max_w \frac{w^T L w}{w^T w}$$

$$\text{s. t. } Bw = 0$$

The constrained eigenvectors themselves are given [GvMG89] by the eigenvectors of  $PLP$ , where



**Figure 5:** Examples of (top to bottom) fundamental, sparse and eigen-liftings, for the three planar graphs in the top row.

$$P = I - B^T(BB^T)^{-1}B$$

## 6.2 Other Constraints

We now show how to generalize the approach presented in Section 5. We assume a scenario similar to that of Section 4, where each face  $f_j$  comes with a vector of parameters  $a_j$  (e.g. face normal). We consider the following problem:

$$\begin{aligned} \min_{q_i \in Q} \sum_i^{N_v} D(q_i, S) \\ \text{s.t. } F(q_i, a_j) = 0, \quad \forall (i, j) \in C \\ G(a_j) = 0, \quad \forall j \end{aligned} \quad (7)$$

where  $D$  is a distance measure. Namely, we wish to minimize the sum of distances of the  $q_i$ 's from a surface  $S$ , while satisfying a corner constraint  $F$  and a face constraint  $G$ . For example, the planarization problem imposes a corner constraint that each vertex be in the planes of its faces, and a face constraint that each normal has unit norm.  $F$  and  $G$  can also be an entire vector of constraints.

We can write (7) in the penalty form:

$$\begin{aligned} \min_{q_i \in Q} \mu \sum_i^{N_v} D(q_i, S) + (1 - \mu) \sum_{(i, j) \in C} F^2(q_i, a_j) \\ \text{s.t. } G(a_j) = 0, \quad \forall j \end{aligned}$$

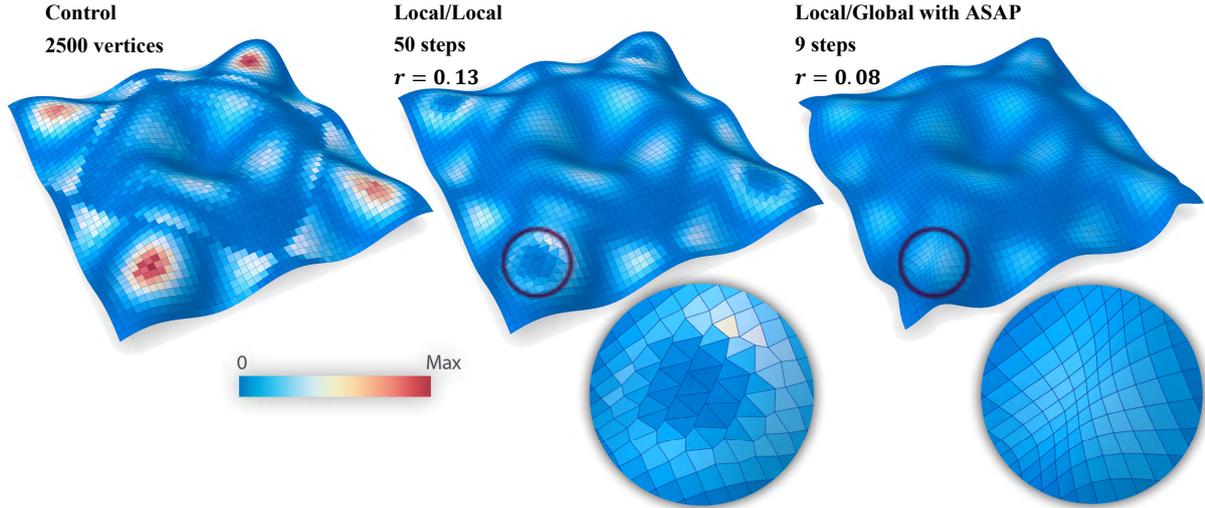
where the squaring of  $F$  can be dropped if  $F$  is known to be positive. Fixing the  $q_i$ 's, we are left with the separate problems

$$\begin{aligned} \min_{a_j} F^2(q_i, a_j) = 0 \\ \text{s.t. } G(a_j) = 0, \quad \forall j \end{aligned}$$

Even if these problems are still complicated, they are much smaller and can be solved in parallel. To get the alternating version of the problem, we must fix some of the face parameters. If we fix all of them, then the problem is a local/local type. Otherwise, it is a local/global type. Deciding which parameters to fix is problem-dependent; We can fix as many parameters as we want, as long as the local/global solution approximates the global solution for each  $\mu$ . Although hard constraints become soft in each iteration using the penalty method, they are effectively hard. We may choose however, to explicitly use soft constraints, making them work essentially as regularizers.

As an example, we construct what may be called an As-Rigid-As-Possible (ARAP) or As-Similar-As-Possible (ASAP) planarization. This means that we aim for each solution face to be a rigid or similar copy of the corresponding control face, as much as possible. To this end, we define for each face  $f_j$  the normal  $n_j$  and distance  $d_j$  for planarization as usual. In addition, we define a rigid/similarity transformation  $A_f$  and a translation vector  $T_f$ . Then problem (7) may be written first with hard constraints as:

$$\begin{aligned} \min \sum_{i=1}^{N_v} \|p_i - q_i\|^2 \\ \text{s.t. } \underline{\text{Planarity}} \\ n_j \cdot q_i + d_j = 0, \quad \forall (i, j) \in C \\ \|n_j\|^2 = 1 \\ \underline{\text{Rigidity/Similarity}} \\ A_j p_i + T_j = q_i \\ A_j^T A_j = sI \end{aligned}$$



**Figure 6:** *As-similar-as-possible planarization. The control mesh (left) is planarized by the regular L/L scheme (center) and the L/G scheme with ASAP regularization (right). ASAP results in much less distortion of the faces, as indicated by the value of  $r$  - the reduction in the non-planarity measure.*

where  $s = 1$  in the case of rigidity. Of course, the constraints are inconsistent in general, hence we convert the rigidity/similarity from a hard constraint to a regularization term, and write it in penalty form, resulting in

$$\begin{aligned} \min \quad & \mu \left( \sum_{i=1}^{N_V} \|p_i - q_i\|^2 + \lambda \sum_{(i,j) \in \mathcal{C}} \|A_j p_i + T_j - q_i\|^2 \right) + \\ & + (1 - \mu) \sum_{(i,j) \in \mathcal{C}} (n_j \cdot q_i + d_j)^2 \\ \text{s.t.} \quad & \|n_j\|^2 = 1 \\ & A_j^T A_j = sI \end{aligned}$$

Deriving the local/global form is now simple; By fixing  $q_i$ , the rest of the variables can be found locally as follows: each  $(n_j, d_j)$  may be computed using the same method as in Section 5 and each  $(A_j, T_j)$  may be computed via the local problem:

$$\begin{aligned} \min \quad & \|A_j p_i + T_j - q_i\|^2 \\ \text{s.t.} \quad & A_j^T A_j = sI \end{aligned}$$

which is nothing but the orthogonal Procrustes problem, the solution of which can be found using Singular Value Decomposition (SVD) [ZG07]. For the global step, we fix only the  $n_j$  and  $A_j$  and are left with a constrained linear least squares problem for the rest of the variables.

As before, an L/L scheme is also possible. However, our experiments show that for this problem, an L/L scheme does not perform well, and in fact, the results are very similar to those of Algorithm **Planarize** (meaning that the similarity/rigidity constraints are not satisfied well). On the other hand, the L/G scheme with ASAP/ARAP regularization reduce the visual artifacts considerably for the same measure of planarity, compared to simple planarization. We compare the results of applying two local schemes to

the mesh in Fig. 6: The L/L scheme, with point-to-point distance and the L/G scheme with ASAP regularization *without* a proximity term. The results show that in problematic areas, the L/L scheme degenerates the quads into triangles, while ASAP regularization preserves the quads, sometimes making them smaller. This better preserves the structure of quad meshes created as conjugate line networks, which may be damaged otherwise.

As a final note, we mention two useful constraints that can be formulated to fit our approach. First, circularity [LPH\*06], where the vertices of each face lie on a circle, can be obtained by adding a center and a radius as additional parameters for each face. Second, we can constrain quad faces to be parallelograms using two additional spanning vectors per face.

## 7. Conclusions and Discussion

We have presented a methodology for efficiently computing and manipulating polyhedral meshes. It removes the serious computational bottleneck of finding an approximating polyhedral mesh to a given reference mesh, while incorporating a variety of constraints. Not only is it more efficient, but it is significantly easier to implement than a dedicated optimizer. Our experiments show that for inputs which are close to polyhedral, such as those generated by many meshing methods, all optimization methods (ours and others) reach very similar results. Hence, these meshing schemes can only benefit by incorporating our much faster optimization procedure.

Preserving the general “look” of a given mesh, e.g. conjugate lines, when converting it to polyhedral form, is quite important in practical applications. We have shown that this may be achieved somewhat when imposing an “As-Similar-As-Possible” constraint on the solution. We wonder if there are other ways to achieve a similar result, also amenable to our techniques.

We would like to explore further the linear theory of optimal liftings. Interesting questions are the dimension of the

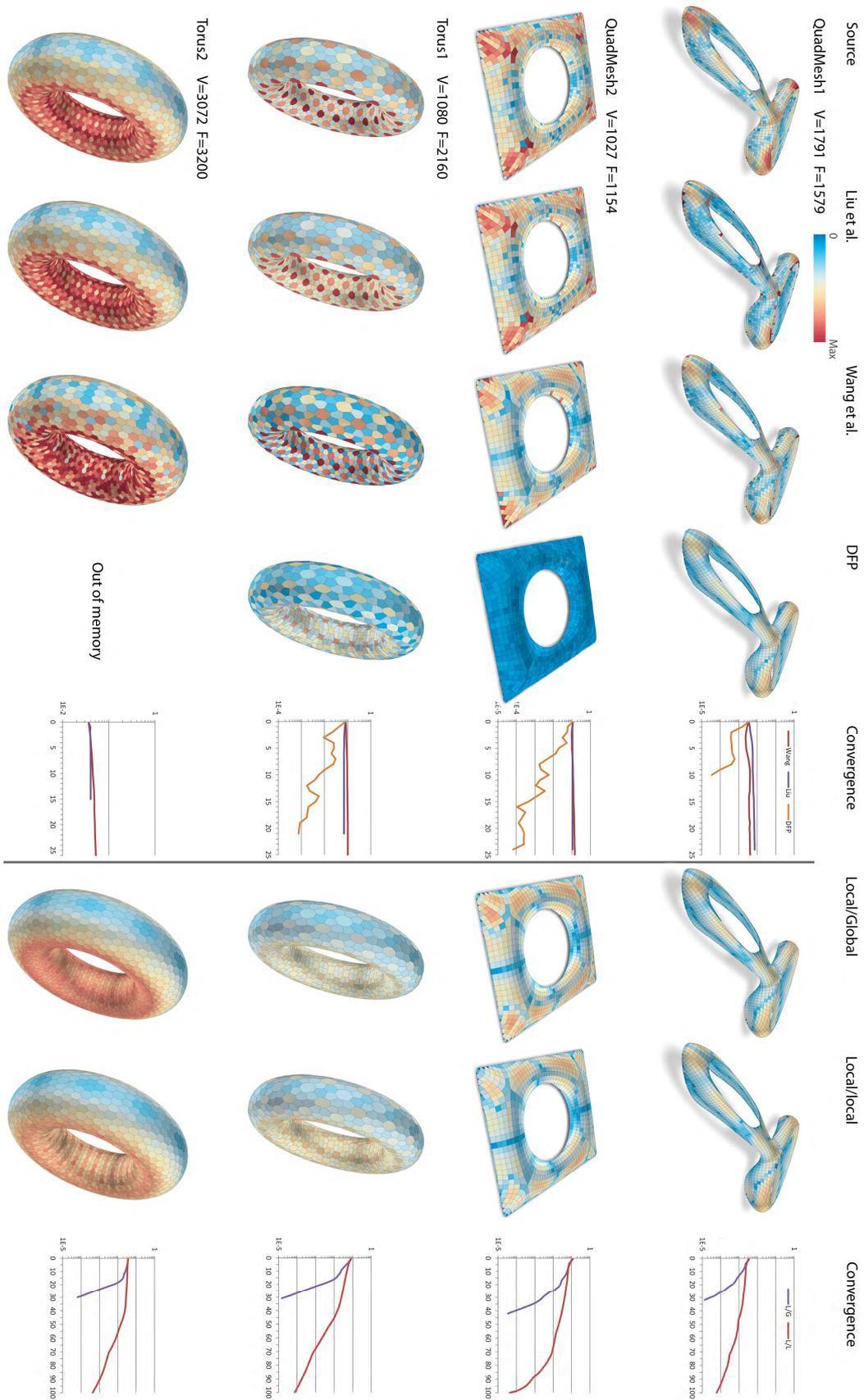
space of liftings as a function of the mesh structure and meaningful bases for this space. We would also like to explore the other linear spaces of polyhedral meshes.

**Acknowledgements.** We thank Johannes Wallner and Alexander Schiffner for providing us the real world meshes in Fig. 7 (top two meshes). We would also like to thank Philippe Block and Lorenz Lachauer for introducing us to the optimal lifting problem and the meshes in Fig. 4.

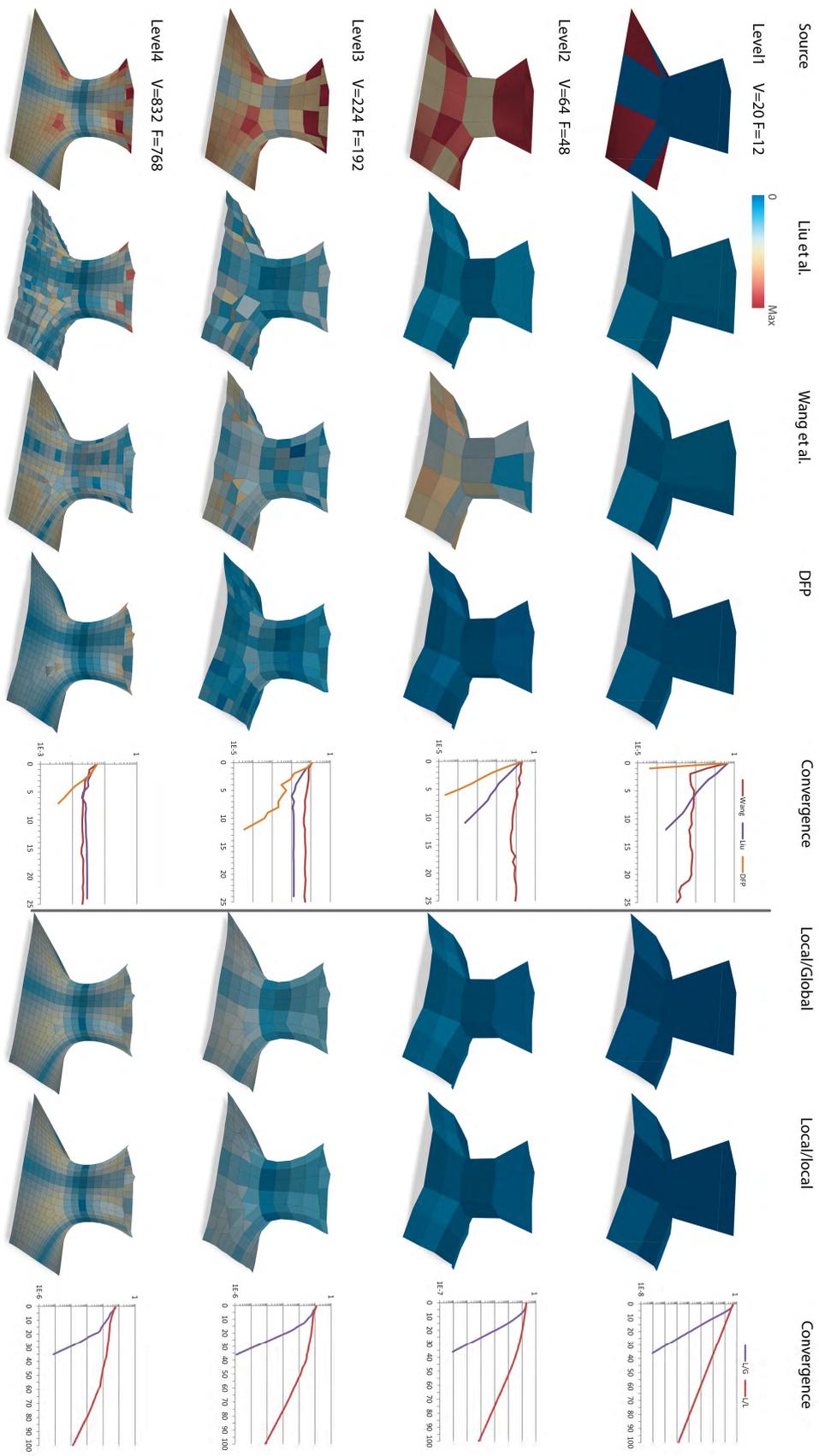
C. Gotsman's work was partially supported by the B. and G. Greenberg Research Fund (Ottawa).

## References

- [AW11] ALEXA M., WARDETZKY M.: Discrete Laplacians on general polygonal meshes. *Proc. SIGGRAPH*, 102:1–102:10 (2006).
- [BNW06] BYRD R. H., NOCEDAL J., WALTZ R. A.: KNITRO: An integrated package for nonlinear optimization. In *Large Scale Nonlinear Optimization, 3559, 2006*, Springer Verlag, 35–59.
- [BDS\*12] BOUAZIZ S., DEUSS M., SCHWARTZBURG Y., WEISE T., PAULY M.: Shape-Up: Shaping Discrete Geometry with Projections. *Comp. Graph. Forum* 31, 5 (August 2012).
- [OMP] OPENMP. API specification for parallel programming. <http://openmp.org/wp/>
- [CM92] CHEN Y., MEDIONI G.: Object modeling by registration of multiple range images. *Image Vision Comput.* 10, 3 (Apr. 1992), 145–155.
- [CSAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. *Proc. SIGGRAPH*, 905–914 (2004).
- [CW07] CUTLER B., WHITING E.: Constrained planar remeshing for architecture. In *Proceedings of Graphics Interface 2007* (New York, NY, USA, 2007), GI '07, ACM, pp. 11–18.
- [DPW11] DENG B., POTTMANN H., WALLNER J.: Functional webs for freeform architecture. *Computer Graphics Forum* 30(5):1369–1378 (2011).
- [GGM89] GANDER W., GOLUB G.H., VON MATT U.: A constrained eigenvalue problem. In *Linear Algebra Appl.* 114–115, pp. 815–839, (1989).
- [GJ10] GUENNEBAUD G., JACOB B., *ET AL.*: Eigen v3. <http://eigen.tuxfamily.org>.
- [HOF10] HOFFMANN T.: On local deformations of planar quad meshes. In *Proc. of ICMS (Berlin, Heidelberg, 2010)*, ICMS'10, Springer-Verlag, pp. 167–169.
- [LPH\*06] LIU, Y., POTTMANN, H., WALLNER, J., YANG, Y.-L., WANG W.: Geometric modeling with conical meshes and developable surfaces. *ACM Trans. Graphics* 25, 3 (July 2006) 681–689.
- [LZX\*08] LIU L. ZHANG L., XU Y., GOTSMAN C., GORTLER S.: *A local/global approach to mesh parameterization*. In *Proc. of SGP (Aire-la-Ville, Switzerland, Switzerland, 2008)*, SGP '08, Eurographics Association, pp. 1495–1504.
- [NW99] NOCEDAL J., WRIGHT S.: *Numerical optimization*. Springer, Aug. 2000.
- [RIC96] RICHTER-GEBERT J. *Realization Spaces of Polytopes*, Vol. 1643 of Lecture Notes in Mathematics. Springer (1996).
- [SA07] SORKINE O., ALEXA M.: *As-rigid-as-possible surface modeling*. In *Proc. of SGP (Aire-la-Ville, Switzerland, Switzerland, 2007)*, SGP '07, Eurographics Association, pp. 109–116. 1
- [TUT63] TUTTE W. T.: *How to draw a graph*. *Proc. London Math. Soc.* 13(1):743–767 (1963).
- [WLY\*08] WANG W., LIU Y., YAN D., CHAN B., LING R., SUN F.: *Hexagonal meshes with planar faces*. Technical Report CS-2008-13, Hong Kong Univ (2008).
- [YYPM11] YANG Y.-L., YANG Y.-J., POTTMANN H., MITRA N. J.: *Shape space exploration of constrained meshes*. *ACM Transactions on Graphics* 30, 6, 124:1–124:12 (2011).
- [ZSW11] ZADRAVEC M., SCHIFTNER A., WALLNER J.: *Designing quad-dominant meshes with planar faces*. *Computer Graphics Forum* 29, 5 (2010), 1671–1679.
- [ZG07] ZHU Y., GORTLER S. J.: *3D deformation using moving least squares*. Tech. Rep. TR-10-07, Harvard Univeristy (2007).



**Figure 7:** Planarization results and convergence using a global algorithm with different types of constraints mentioned in this paper and the local schemes derived from DFP formulation.



**Figure 8:** More planarization results and convergence using a global algorithm with different types of constraints mentioned in this paper and the local schemes derived from DFP formulation. These meshes were generated by applying the Catmull-Clark subdivision scheme to the top mesh.

