Designing Patterns using Triangle-Quad Hybrid Meshes

CHI-HAN PENG, KAUST
HELMUT POTTMANN, TU Wien
PETER WONKA, KAUST

We present a framework to generate mesh patterns that consist of a *hybrid* of both triangles and quads. Given a 3D surface, the generated patterns fit the surface boundaries and curvatures. Such regular and near regular triangle-quad hybrid meshes provide two key advantages: first, novel-looking polygonal patterns achieved by mixing different arrangements of triangles and quads together; second, a finer discretization of angle deficits than utilizing triangles or quads alone. Users have controls over the generated patterns in global and local levels. We demonstrate applications of our approach in architectural geometry and pattern design on surfaces.

CCS Concepts: • **Computing methodologies** → **Mesh models**; **Mesh geometry models**;

Additional Key Words and Phrases: Meshes, Polygonal Patterns, Pattern Design, Geometric Modeling

ACM Reference Format:
Chi-Han Peng, Helmut Pottmann, and Peter Wonka. 2018. Designing Patterns using Triangle-Quad Hybrid Meshes. *ACM Trans. Graph.* 37, 4, Article 1 (August 2018), 14 pages. https://doi.org/10.1145/3197517.3201306

1 INTRODUCTION

The study of tilings and patterns has various applications, e.g. in architecture, industrial design, and art. In this work we study patterns that are generated from triangle- and quad-shaped tiles. One motivation for our work is that there is a large body of research studying the meshing of surfaces using triangle, quad, or hex meshes. In addition, there are also quad-dominant meshes, that mainly consist of quads with a few isolated triangles. In this paper, we would like to explore a richer class of designs that are not confined to be either triangle or quad(-dominant) meshes, but a *hybrid* of both. See Fig. 1 for a novel architectural mesh design enabled by our work.

To start our investigation, we first look at meshes in the plane consisting of regular triangles and squares. In this context, the tiling of the infinite Euclidean plane has been studied for various tile sets, including tile sets using quads and triangles [Grunbaum and Shephard 2016]). However, an open question is how to tile a finite domain with a given boundary using squares and regular triangles. As an important building block of our work we analyze this tiling problem and propose an optimization algorithm to compute a single feasible tiling, enumerate all possible tilings within the given region, or select a particular tiling by modeling an objective function that includes aesthetic criteria (Section 3.3).

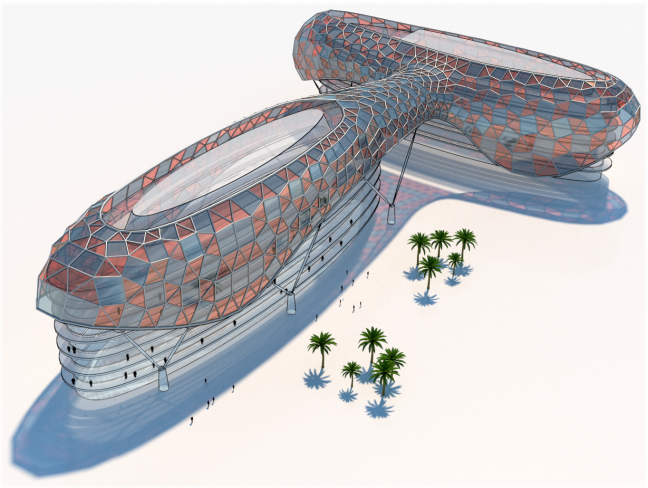


Fig. 1. A triangle-quad hybrid mesh was used to design an alternative outer skin of the Yas Island Marina hotel. The design of regular and near regular triangle-quad patterns on surfaces is a difficult task that is tackled by the computational framework in this paper.

To tackle the design problem in 3D, we combine multiple 2D solutions. We use a graph structure called *patch graph*, to encode the boundaries of individual 2D regular hybrid meshes and cross-patch consistency constraints (Section 4.1). The individual 2D solutions can then be deformed and glued together to yield a 3D design. In some cases, interesting designs can also be generated without deforming the individual 2D solutions, giving rise to an interesting generalization of Lobel frames [Lobel 2004]. Patch graphs can be generated from scratch, by a subdivision scheme, or designed on a given reference surface using our proposed user-interface (Section 4.2).

In Section 5, we showcase various designs enabled by our framework to demonstrate that hybrid meshes are an interesting complement to existing triangle and quad-based meshes. Some advantages of hybrid meshes for fabrication are a finer accuracy of the discretization of angle deficits at a vertex compared to triangle and quad meshes as well as the easier planarization compared to quad meshes. The main contributions of this paper are summarized as follows:

- Introducing the concept of *triangle-quad hybrid meshes*, with which we discovered novel polygonal patterns with a distinct look that cannot be derived from existing triangle/hex or quad meshes.

Authors' addresses: Chi-Han Peng, KAUST, chihan.peng@kaust.edu.sa; Helmut Pottmann, TU Wien, pottmann@geometrie.tuwien.ac.at; Peter Wonka, KAUST, peter.wonka@kaust.edu.sa.

0730-0301/2018/8-ART1 $15.00
https://doi.org/10.1145/3197517.3201306

- An IP optimization-based method that can exhaustively enumerate 2D hybrid mesh patterns consisting of equilateral triangles and squares for a given 2D boundary.
- An extension to 3D that enables the computation of compatible 2D solutions that can be deformed and glued together to obtain a 3D design.
- A user-interface for designing a patch segmentation on a given 3D surface.
- Demonstrating novel and high-quality freeform architecture designs and artistic patterns created by our method.

2 RELATED WORK

Tiling and Tessellation. The essence of a tiling is that the tiles cover the domain without overlap. In contrast, a *packing* requires that the tiles do not overlap, but it does not require the complete domain to be covered. There are multiple examples of packing problems in computer graphics, e.g. decorative mosaics [Hausner 2001; Kim and Pellacini 2002], artistic packing layouts [Reinert et al. 2013], texture atlases [Lévy et al. 2002], and packing on surfaces [Chen et al. 2017; Hu et al. 2016; Schumacher et al. 2016]. Another variation of the problem is to allow tiles in a tiling to deform. Example applications in computer graphics are layouts for urban environments and floorplans [Peng et al. 2014; Yang et al. 2013]. A focus has been on how to design the tile set that would best tile the given domain. An iconic example is the "Escherization" problem - how to design isohedral or dihedral tile sets that closely resemble user-supplied goal shapes to tile the infinite Euclidean plane [Kaplan and Salesin 2000, 2004]. In this paper, our goal is to find ways to completely tile a domain with a given boundary with a predefined tile set - equilateral triangles and squares.

In computer graphics, tiling-based approaches also play a key role in texture synthesis [Cohen et al. 2003] and blue noise generation [Ahmed et al. 2016; Kopf et al. 2006].

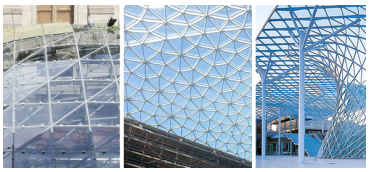
We drew inspiration from studies about tilings and pattens (see [Grunbaum and Shephard 2016] for a comprehensive survey). Those generated a large variety of polygonal patterns, including triangle-quad hybrid meshes, with a focus on aesthetic qualities such as symmetry and congruence. However, a key limitation is that they typically aim to tessellate the infinite Euclidean plane, thus preventing direct applications on surfaces where the boundaries and curvature need to be respected.

Crystallography. Triangle-quad hybrid meshes, when generated in a rotationally symmetric manner, may resemble *quasicrystal* patterns. In the studies about symmetry structure of quasiperiodic tiling classes ([Hermisson et al. 2002] and chapter 2 in [Suck et al. 2002]), square-triangle tilings are shown as results. Our method, having the ability to exhaustively enumerate hybrid mesh patterns for a given 2D boundary that is not restricted to be symmetric, may be useful for crystallography researchers to discover new patterns.

Surface meshing. Surface meshing methods (we recommend the book [Botsch et al. 2010] for a broad survey) can be categorized by the types of faces being used. One type of meshes has no restriction on the face type. Examples include centroidal Voronoi and power diagrams [Alliez et al. 2005; Xin et al. 2016] and surface partition cells [Cohen-Steiner et al. 2004; Sander et al. 2001]. Meshes consisting of a single face type are pure triangle meshes, quad meshes, or hex meshes. In quad meshing, there are methods that solve tessellation with boundary constraints using integer linear programming (ILP) [Marcias et al. 2015; Takayama et al. 2014]. To make meshing easier, it is advantageous to mix the dominant face type with others, e.g., quad-dominant meshes mainly consist of quads with a few isolated triangles (see [Bommes et al. 2013] for a survey). Meshes can also be subdivided to give rise to a richer set of patterns, similar to semi-regular tilings of the Euclidean plane. Examples include predefined mixed-face types patterns derived from existing triangle/hex or quad meshes [Jiang et al. 2015; Vaxman et al. 2017] and non-convex hexagonal faces [Li et al. 2015]. Picking up this direction, our goal is to devise a method that can generate meshes from scratch using triangles and quads with equal preference. The results are novel mesh patterns that could no longer be recognized as being triangular/hexagonal or quadrilateral, but yet still have a semi-regular structure consisting of two types of faces.

Subdivision surfaces. Subdivision operators that unify triangular and quadrilateral subdivision schemes exist [Schaefer and Warren 2005; Stam and Loop 2003]. Our method can be used to generate base meshes that consist of both triangle and quad patches.

Architecture. Several steel/glass constructions in contemporary architecture are based on hybrid triangle/quad meshes. There, planarity of quads is important and is facilitated by the presence of triangles (see [Pottmann et al. 2015]). We mention three projects (see inset, left to right): The roof of the Islamic Art Exhibition in the Louvre is based on a simple repetitive pattern. The Yas Mall in Abu Dhabi exhibits an ornamental pattern of triangles and quads with a strong dominance of triangles. At the Fiera Milano, we see a composition of triangle and quad meshes (in nearly flat regions) with randomly appearing splittings of non-planar quads into triangles. A square has more than twice the area than a triangle of the same edge length. Their use as panels reduces material usage and weight of the support structure that follows the edges. As meshes from quads close to squares (or rectangles) discretize principal parameterizations, they lack degrees of freedom for many practical scenarios. The presence of triangles not only increases the chance to planarize the quads, but also stiffens the entire structure. Current solutions are mostly derived from other meshes by simple edge insertion/deletion rules or by cutting out a subset of tri-quad patterns in the infinite Euclidean plane to fit the 2D or 3D surface. Then the mesh patterns would not respect the boundaries (e.g., the Yas Mall). In contrast, our method has the goal of computing hybrid meshes from a given boundary constraint at the start.



3 REGULAR HYBRID MESHES IN THE PLANE

In this section, we begin with the definition of regular hybrid meshes in the plane and explore some of their properties. In Section 3.2, we introduce a discrete coordinate system to encode the vertices

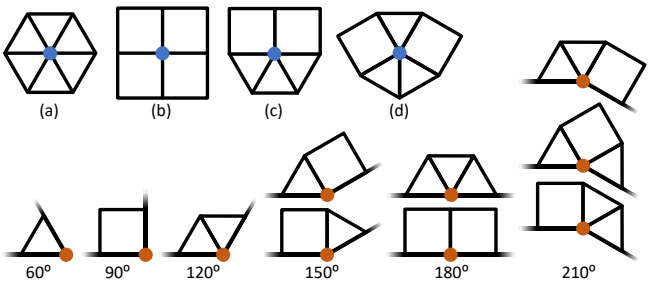


Fig. 2. Top row (a)-(d): the four configurations of a regular interior vertex. (d) is of particular interest to us and is denoted as the "butterfly" configuration. Bottom row: configurations of boundary vertices of various angles.

in a hybrid mesh, which is a key component for our planar regular hybrid mesh generation method (Section 3.3).

A *regular hybrid mesh in 2D* is a tessellation of a connected part of the Euclidean plane E^2 with equilateral triangles and squares. The *configuration* of a vertex is the sequence of types of its adjacent faces (either triangle or quad) in the counter-clockwise order. For interior vertices, it starts at the longest consecutive sequence of triangles. Two configurations matched by rotational symmetry are considered the same. For boundary vertices, there exists exactly one way to encode the face types.

There are exactly four configurations for an interior vertex (see Fig. 2 (a)-(d)). Boundary vertices can be distinguished by their boundary angle, which can be any integer multiple of $30^\circ \in [60, 330]$. As an example, we show all configurations for boundary angles $\in [60, 210]$ (Fig. 2 bottom).

In a regular hybrid mesh, there are twelve possible directions for every half-edge [1]. By arbitrarily fixing one half-edge direction, we can encode each half-edge's direction as an integer $\in [0, 11]$.

3.1 Subdivision scheme

To obtain some insights about regular hybrid meshes, we introduce a subdivision scheme as an auxiliary structure. The scheme, shown in Fig. 3 (a), first inserts all edge midpoints as new vertices. For a triangle, each edge midpoint is connected to the opposite vertex and the two other edge midpoints. For a quad, each edge midpoint is connected to the two vertices and midpoint of the opposite edge. Assuming that the mesh edges are of length 2, they are categorized and denoted by their Euclidean lengths as *1-edge*, *2-edge*, $\sqrt{3}$*-edge*, and $\sqrt{5}$*-edge*. Mesh edges, which are 2-edges, can be divided into two consecutive 1-edges if needed.

In this subdivision scheme, a 60° angle inside a triangle is divided into two 30° angles, and a 90° angle inside a quad is divided into three angles, which are not exactly 30°. This causes some distortions because the $\sqrt{5}$-edges inserted into quads are no longer aligned with the original twelve directions. However, we can still assign to a $\sqrt{5}$-edge the direction label of the closest among the twelve directions. The deviation angle between these two directions is a small positive or negative number. Therefore, we distinguish between a $\sqrt{5}$-edge-a and a $\sqrt{5}$-edge-b respectively.

[1] We assume the half-edges of a face circulate in the counter-clockwise order.

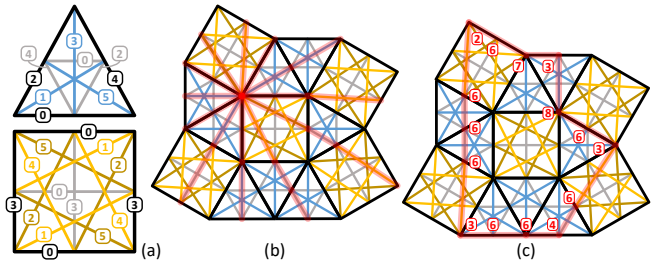


Fig. 3. (a) Subdividing triangles and quads. We show one direction label per edge (the smaller one for the two half-edges). Original 2-edges are shown in black and newly inserted 1-edges and 2-edges are shown in gray. $\sqrt{3}$-edges are shown in blue, $\sqrt{5}$-edges-a in darker brown, and $\sqrt{5}$-edges-b in lighter brown. (b) The main paths at the twelve directions from a vertex. Observe that a main path is not necessarily a straight line in the Euclidean plane. (c) A patch (see Section 4.1). The c-indices of the vertices and edge mid-points along the patch boundary are shown in red.

We define a *main path* (see Fig. 3 (b)) in a subdivided regular hybrid mesh as a connected sequence of half-edges with the same direction label. Note that a main path does not necessarily lie on a straight line. We state an important observation as follows:

Lemma 3.1. *Every interior half-edge in a regular hybrid mesh is part of a main path that goes from one boundary point to another. A boundary point is either a boundary vertex or a midpoint of a boundary edge.*

3.2 Discrete coordinate system

Our hybrid mesh generation method, detailed in Section 3.3, requires a discrete coordinate system to uniquely encode every vertex in a regular hybrid mesh with a fixed-length sequence of integers. To make the coordinates unique, we need to identify an arbitrarily chosen origin vertex and orient the mesh in a Cartesian system (e.g. by aligning a half-edge to the x-axis).

The main purpose of this coordinate system is to identify and enumerate vertices and edges of possible regular hybrid meshes with a given boundary. Specifically, it should facilitate efficient checking if two vertices are on a main path (needed by the coordinate test described in Sec. 3.3).

The 4D system. Assuming the edges of a regular hybrid mesh in E^2 are of length 2, direction vectors of edges can only have x- and y-coordinates from the following set: $\{0, \pm 1, \pm 2, \pm\sqrt{3}\}$. Therefore, each vertex in the mesh must have a coordinate vector v of the form

$$v = (A + B\sqrt{3}, C + D\sqrt{3}), \tag{1}$$

with *integers* (A, B, C, D), called *discrete 4D coordinates* of v. By irrationality of $\sqrt{3}$, these integer coordinates are unique, and uniquely define the Cartesian coordinates of v. See Fig. 4 (a).

Can any four integers arise as 4D coordinates? The possible edge direction vectors are $(2, 0, 0, 0)$, $(0, 1, 1, 0)$, $(1, 0, 0, 1)$, $(0, 0, 2, 0)$, $(-1, 0, 0, 1)$, $(0, -1, 1, 0)$ and the ones obtained by multiplication with -1. For all of them, $A + D$ and $B + C$ take values in $\{0, 2, -2\}$. As any vertex coordinate is a linear combination of these vectors, only

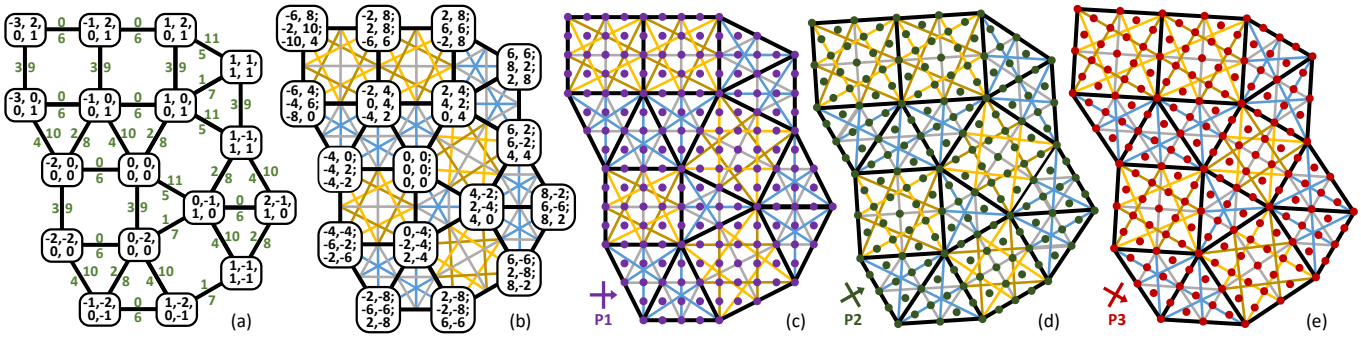


Fig. 4. (a) Discrete vertex coordinates in the 4D system and direction labels of the half-edges (shown in green). (b) Six-integer coordinates of vertices. (c)-(e): The three projections of the mesh. Observe that each vertex's six-integer coordinate is the concatenation of its coordinates in the three 2D projections.

4D coordinates with even values of $A + D$ and $B + C$ can arise. This poses no problem in the algorithm.

The 4D system does not facilitate efficient checking if two vertices are on a main path. Recall that, in E^2, a main path may not be exactly straight due to the presence of the $\sqrt{5}$-edges. In $4D$, a main path isn't straight either, and in addition, the 4D embedding, M^4, of a regular hybrid mesh, $M \subset E^2$, does not lie in a plane anymore. In general, it spans 4D space. We can now apply appropriate parallel projections from 4D into 2D which map main paths onto straight lines. For that, we need three parallel projections (linear maps), discussed next.

Projection 1. We want to design parallel projection, $P_1 : (A, B, C, D) \mapsto (x_1, y_1)$, so that the main paths in the 0-, 3-, 6-, or 9-th direction (i.e., those nearly parallel to the x- or y-axis of the Cartesian system in E^2) become straight. This is achieved as follows: Edge vectors which are parallel to the x-axis and of length 2 (4D coordinates: $(2, 0, 0, 0)$) or of length $\sqrt{3}$ (4D coordinates: $(0, 1, 0, 0)$) are mapped to the same vector; we choose it to be $(4, 0)$ to stay entirely with small integer coordinates. We map vectors of length 2 or of length $\sqrt{3}$ in the y-direction to $(0, 4)$. Hence, P_1 has to satisfy:

$$(2, 0, 0, 0) \mapsto (4, 0),\ (0, 1, 0, 0) \mapsto (4, 0),$$
$$(0, 0, 2, 0) \mapsto (0, 4),\ (0, 0, 0, 1) \mapsto (0, 4).$$

As we know the images of four independent vectors, the linear map P_1 is uniquely defined:

$$P_1 : (A, B, C, D) \mapsto (x_1, y_1) = (2A + 4B, 2C + 4D). \tag{2}$$

Remarkably, P_1 maps the $\sqrt{5}$-edges which are nearly parallel to the x- or y-axis to vectors $(\pm 5, 0)$ or $(0, \pm 5)$, respectively. By symmetry, it is sufficient to prove this for one of these $\sqrt{5}$-edges. In E^2, it shall be represented by $(\sqrt{3}, 1) + \frac{1}{2}(1, -\sqrt{3})$; its P_1-image is indeed $(4, 2) + \frac{1}{2}(2, -4) = (5, 0)$. Hence, P_1 straightens exactly the main paths in directions 0, 3, 6, and 9.

Projection 2. Now we straighten the main paths in directions 1, 4, 7, and 10 (i.e., those that are inclined against the x-axis by nearly 30° or -60° counter-clockwise). In short, the second projection, P_2, shall map 2-edge $(0, 1, 1, 0)$ and $\sqrt{3}$-edge $(3/2, 0, 0, 1/2)$ to $(4, 0)$ and those obtained by a 90° rotation to $(0, 4)$, which yields:

$$P_2 : (x_2, y_2) = (2A + 3B + C + 2D, -A - 2B + 2C + 3D). \tag{3}$$

Again, the $\sqrt{5}$-edges get mapped to $(\pm 5, 0)$ or $(0, \pm 5)$.

Projection 3. Finally, we straighten the main paths in directions 2, 5, 8, and 11 (i.e., those that are inclined against the x-axis by nearly 30° or -60° clockwise). This requires:

$$(0, 1, -1, 0) \mapsto (4, 0),\ (3/2, 0, 0, -1/2) \mapsto (4, 0),$$
$$(1, 0, 0, 1) \mapsto (0, 4),\ (0, 1/2, 3/2, 0) \mapsto (0, 4),$$

and is achieved via

$$P_3 : (x_3, y_3) = (2A + 3B - C - 2D, A + 2B + 2C + 3D). \tag{4}$$

An example of these three projections is shown in Fig. 4 (c)-(e). Note that three 2-dimensional linear images of 4D-space cannot be independent. The images under the three projections are related by the equations:

$$x_1 + y_2 - y_3 = 0,\ y_1 - x_2 + x_3 = 0. \tag{5}$$

If just four of the six coordinates $(x_1, y_1, x_2, y_2, x_3, y_3)$ are known, we can retrieve the $4D$-coordinates and hence the Cartesian coordinates in E^2. Hence, the inversion formulae are not unique. One version of them is,

$$\begin{aligned} A &= (-3x_1 + 2x_2 + 2x_3)/2, \\ B &= (2x_1 - x_2 - x_3)/2, \\ C &= (-3y_1 + 2y_2 + 2y_3)/2, \\ D &= (2y_1 - y_2 - y_3)/2, \end{aligned} \tag{6}$$

others follow with help of (5).

Note that this projection method is very similar to the approach in Descriptive Geometry: There, a point (x, y, z) in 3D is mapped to at least two views, usually top view (x, y) and front view (y, z), and of course, these views are not independent. An object may appear simpler or a construction be easier in a special view. For example, instead of top and front view, one will use another side view. This is also the case here. We choose the projections so that selected non-straight paths become straight.

By concatenating the three 2D projections, (x_1, y_1), (x_2, y_2), and (x_3, y_3), into single six-integer vectors, we arrive at our solution of edge encodings, summarized in Table 1. For brevity, only the first six directions are listed and encodings for 1-edges are omitted since they are half of the encodings of the 2-edges in the same directions. In Fig. 4 (b), we show the six-integer coordinates of vertices using this system. Note that any main path becomes a straight line parallel

2-edge	$\sqrt{3}$-edge	$\sqrt{5}$-edge-a	$\sqrt{5}$-edge-b
(4,0; 4,-2; 4,2)	(4,0; 3,-2; 3,2)	(5,0; 4,-2; 4,3)	(5,0; 4,-3; 4,2)
(4,2; 4,0; 2,4)	(3,2; 4,0; 2,3)	(4,3; 5,0; 2,4)	(4,2; 5,0; 3,4)
(2,4; 4,2; 0,4)	(2,3; 3,2; 0,4)	(2,4; 4,3; 0,5)	(3,4; 4,2; 0,5)
(0,4; 2,4; -2,4)	(0,4; 2,3; -2,3)	(0,5; 2,4; -3,4)	(0,5; 3,4; -2,4)
(-2,4; 0,4; -4,2)	(-2,3; 0,4; -3,2)	(-3,4; 0,5; -4,2)	(-2,4; 0,5; -4,3)
(-4,2; -2,4; -4,0)	(-3,2; -2,3; -4,0)	(-4,2; -3,4; -5,0)	(-4,3; -2,4; -5,0)

Table 1. Six-integer coordinates of edge vectors.

to a coordinate axis in one of the three projections. Therefore, the six-integer coordinates facilitate fast checking if two vertices could lie on a main path as follows. Two vertices are on the same main path in the 0-th / 6-th direction only if the second of the six coordinates are the same. Similarly, they are on the main path in the 1-th / 7-th, 2-th / 8-th, 3-th / 9-th, 4-th / 10-th, or 5-th / 11-th direction only if the fourth, fifth, first, third, or sixth of their coordinates are the same.

3.3 Hybrid mesh generation in a given boundary

We describe a method that can *exhaustively* enumerate regular hybrid meshes in a given 2D boundary, B. Recall that, all edges in a regular hybrid mesh have the same length and all boundary angles are in multiples of $30^\circ \in [60, 330]$. We use a sequence of non-negative integers to encode B. The integers denote the boundary angles in multiples of 30° at every boundary vertices in counter-clockwise order. For example, the patch boundary of the mesh in Fig. 4 is encoded as $(4, 5, 5, 4, 7, 4, 5, 6, 3, 5, 7, 5)$, starting at the boundary vertex in the bottom-left corner. A boundary description is unique up to a cyclic permutation.

The main steps of our algorithm are summarized as follows:

(1) *Initialization.* Given a boundary description as input, compute an embedding in E^2 and six-integer coordinates of boundary vertices and midpoints of boundary edges. See Fig. 5 (a).
(2) *Edge enumeration.* Enumerate a superset S of potential edge placements within the boundary. This is based on *main paths* according to Lemma 3.1 and integer programming (IP).
(3) *Mesh generation.* Solve another IP problem to compute a regular hybrid mesh by selecting edges in S.
(4) *Complete enumeration.* To exhaustively enumerate all possible solutions, we can run IP multiple times, each time banning all previously retrieved solutions.
(5) *Solution optimization.* Additionally, the IP problem can be setup with different objective functions and constraints to retrieve solutions with desired attributes.

In the following, we describe steps (2) to (5) in more detail.

Edge enumeration. Edge enumeration proceeds in two main steps. (1) We enumerate all pairs of boundary points (vertices and edge midpoints) that could lie on a main path inside the boundary, and (2), we enumerate all possible edges between all pairs detected in the first step.

For every pair of boundary points (vertices and edge midpoints), we perform a sequence of tests to determine if they could lie on a main path:

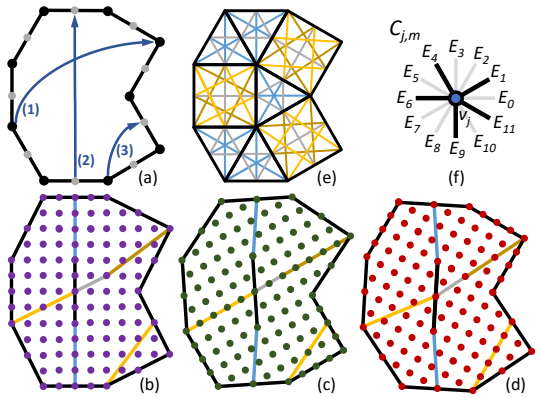


Fig. 5. A closed loop traced by a given boundary description, (4, 5, 3, 8, 3, 5, 4, 5, 5), starting at the bottom-left corner. (a) is the embedding in the Euclidean plane and (b)-(d) are the three projections. (1) to (3) denote three main paths with direction labels 1, 3, and 2. Their six-integer vectors are (10, 6; 12, 0; 6, 10), (0, 12; 6, 10; -6, 10), and (3, 4; 4, 2; 0, 5). (e) is a reference meshing. In (f), we illustrate a configuration variable, $C_{j,m}$, that becomes true if and only if E_1, E_4, E_6, E_9, E_{11} are true and E_0, E_2, E_3, E_5, E_7, E_8, E_{10} are false.

(a) *Coordinate test.* We check if two boundary points could lie on the same main path of a particular direction label, a, by the fast checking rules described at the end of Section 3.2.
(b) *Inside test.* If the boundary is not convex, we need to ensure that the possible main path is inside the boundary. We check that the complete line segment between the pair is inside the boundary in the projection where a main path in direction a is straight.
(c) *Decomposition test.* Here we test whether the line segment between the pair can be decomposed into individual 1-edges, $\sqrt{3}$-edges, $\sqrt{5}$-edges-a, and $\sqrt{5}$-edges-b. We do this by solving the following IP problem. Let V denote the six-integer vector between the two points and let (t_0, t_1, t_2, t_3) denote the numbers of the four edge types,

$$\text{find} \quad t_0, t_1, t_2, t_3 \quad \text{s.t.} \quad \sum_i t_i T_i = V,$$

where T_i is the six-integer vector encoding edge type i in direction a (see Table 1).

Finally, for every quadruple, (t_0, t_1, t_2, t_3), we enumerate all possible edge combinations, (x_0, x_1, x_2, x_3), that appear before some 2-edge, which are:

$$(x_0, x_1, x_2, x_3), x_0 \le t_0 - 2, x_1 \le t_1, x_2 \le t_2, x_3 \le t_3, x_i \ge 0.$$

We now have exhaustively enumerated all possible potential placements of interior 2-edges. Our next task is to find a subset of these 2-edges which together with the boundary 2-edges forms a regular hybrid mesh.

Mesh generation. We formulate the mesh generation step as a linear IP problem. In the previous step, we found potential placements of 2-edges indexed $0 \le i < N_e$. We use Boolean variables E_i to denote the presence of the i-th 2-edge in the mesh. We also enumerate unique potential placements of vertices, denoted as v_j,

$0 \le i < N_v$. This enumeration is implemented using a map data structure with the 4D coordinates of vertices as key. Next we enumerate possible edge configurations around each vertex v_j. For an interior vertex, there are 29 different configurations, because the four basic configurations in Fig. 2 can appear in rotated forms (2, 3, 12, 12 rotations respectively). For a boundary vertex, multiple configurations exist depending on the boundary angle (see Fig. 2 bottom row). We use Boolean variables $C_{j,m}$ to denote the presence of the m-th configuration at v_j. The index set $I_{j,m}$ contains all edge indices corresponding to configuration $C_{j,m}$, and the index set $\bar{I}_{j,m}$ contains indices of all edges adjacent to v_j but not in configuration $C_{j,m}$. For each vertex and each configuration, we formulate the following two configuration constraints. The configuration is active if and only if all corresponding edges are selected and all other edges at v_j are not selected (see Fig. 5 (f)).

$$0 \le \sum_{i \in I_{j,m}} E_i + \sum_{i \in \bar{I}_{j,m}} (1 - E_i) + N_j C_{j,m} < N_j \tag{7}$$

where N_j denotes the number of edge placements adjacent to v_j. We now formulate the complete IP problem as:

$$\begin{aligned}
\text{find} \quad & E_i, 0 \le i < N_e, \quad \text{and} \\
& C_{j,m}, 0 \le j < N_v, 0 \le m < M_j, \\
\text{s.t.} \quad & \text{Equation (7)}, \forall\, v_j, \\
& \sum_m C_{j,m} = 1, \forall \text{ boundary vertices } v_j, \\
& \sum_m C_{j,m} \le 1, \forall \text{ interior vertices } v_j, \\
& \sum_m C_{a,m} - E_i \le 0 \ \text{ and} \\
& \sum_m C_{b,m} - E_i \le 0, \forall\, E_i \text{ with vertices } v_a, v_b
\end{aligned} \tag{8}$$

where M_j denotes the number of configurations at v_j. The second constraint dictates that for every boundary vertex, exactly one of the configurations should appear. The third constraint dictates that for every internal vertex, at most one of the configurations should appear (since many interior vertices will not be part of the mesh). The fourth constraint prevents dangling edges. After solving Problem (8), the set of active E_i form a regular hybrid mesh with the given boundary.

Complete enumeration. We can exhaustively enumerate all possible regular hybrid meshes with the same given boundary by solving Problem (8) repeatedly, each time banning all previously retrieved solutions. This is done as follows. Let Z_i, $0 \le i < N_e$, be the value of E_i of an existing solution. Then, for every existing solution, we have the following constraint:

$$\sum_i (E_i \text{ if } Z_i \text{ is true, or } (1 - E_i) \text{ if } Z_i \text{ is false}) < N_e$$

added to Problem (8).

Configuration optimization. We can augment the IP problem to prioritize or penalize certain types of configurations to appear in the solution. This is done by adding the following objective function to the IP problem:

$$\text{minimize} \quad \sum_j \sum_m \lambda_m C_{j,m}$$

where the λ_m's are the weights for configuration $C_{j,m}$. As shown in Fig. 6 (f)-(g) and Fig. 7 (d)-(e), one useful scenario is to prioritize or penalize the "butterfly" configurations (Fig. 2 (d)) to produce solutions that look more or less fractured.

Symmetry optimization. We can improve the symmetry of the solution in an approximate sense as follows. Given the desired symmetry, e.g., a reflective symmetry or an n-way rotational symmetry with respect to a user-specified center, we synthesize a scalar field F that is invariant under the selected symmetry. This field is generated in two steps. First, we generate a smooth function and then compute a random function using the smooth function values as the seeds. Afterwards, we add the following term to the objective function:

$$\sum_j F_j (\sum_m C_{j,m})$$

where F_j is the value of the scalar field at the location of vertex v_j. As a result, some vertices will be arbitrarily preferred over other vertices. Since this preference respects the selected symmetry, the objective function is typically lower for symmetric configurations. See Fig. 6 (c)-(e) and and Fig. 7 (a)-(c) for examples.

3.4 Approximating 2D boundary curves

We now describe an algorithm to compute a valid regular hybrid mesh boundary from a boundary curve, given as an arbitrary 2D piece-wise linear loop. In short, the former has uniform edge length and angles in multiples of 30° while the latter can have arbitrary edge lengths and angles. A simple greedy algorithm, i.e., starting at one arbitrary boundary vertex and inserting edges one by one, may not create a closed loop in the end. Instead, we formulate the task as an IP optimization problem as follows.

First, we enumerate a superset of potential placements of boundary 2-edges close to the given boundary. For every enumerated edge, we distinguish two half-edges. Our goal now is to find a subset of the half-edges that forms a closed loop that best approximates the given boundary curve. Let Boolean variables $E_{i,j}$ denote the presence of a half-edge from vertex v_i to vertex v_j in the subset. The IP problem is formulated as:

$$\begin{aligned}
\text{find} \quad & E_{i,j}, 0 \le i, j < N_v, \quad \text{such that} \\
\text{min.} \quad & \sum_i Dist_{i,j} E_{i,j}, \\
\text{s.t.} \quad & \sum_i E_{i,j} = 1 \text{ and } \sum_k E_{j,k} = 1, \forall \text{ fixed } v_j, \\
& \sum_i E_{i,j} = \sum_k E_{j,k}, \forall \text{ non-fixed } v_j.
\end{aligned} \tag{9}$$

where N_v is the number of potential boundary vertices enumerated. We distinguish a set of vertices to be *fixed*, i.e., they must appear in the mesh boundary solution, by a distance threshold to the given boundary curve (note that the vertex at $(0, 0)$ is necessarily fixed). $Dist_{i,j}$ is the average distance of half-edge $E_{i,j}$ to the given boundary

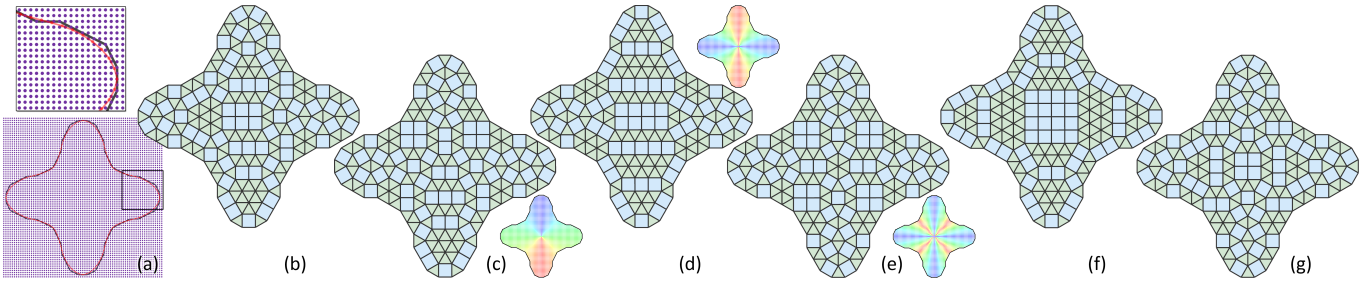


Fig. 6. Generating regular hybrid meshes for a 2D boundary curve inspired by the cross-section of the Lilium Tower by Zaha Hadid Architects. (a) The input boundary curve (red) is approximated by a regular hybrid mesh boundary (black). (b) A regular hybrid mesh solution computed without symmetry constraints, (c) with left-right reflective symmetry constraint, (d) with left-right and top-down reflective symmetry constraints, (e) and with four-way rotational symmetry constraint. The smooth versions of the underlying scalar fields are also shown. (f)-(g): Solutions computed while penalizing (f) or prioritizing (g) the appearances of the "butterfly" configurations. (f) and (g) have 40 and 84 butterfly configurations, respectively.

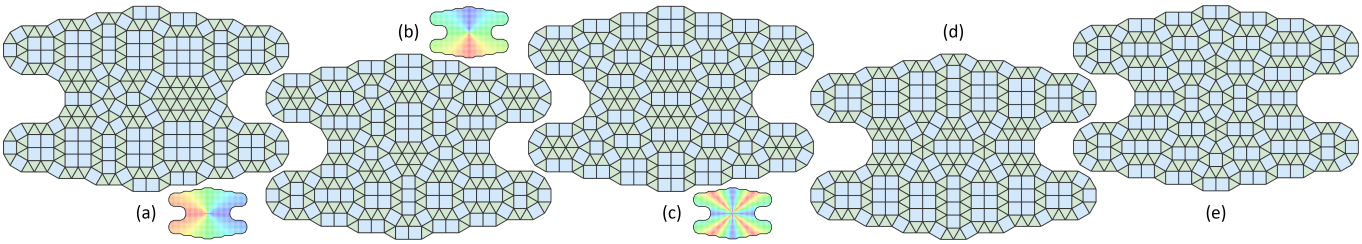


Fig. 7. Regular hybrid meshes for a 2D boundary curve inspired by the Japan National Stadium design by Zaha Hadid Architects. (a) A solution computed with up-down reflective symmetry, (b) with left-right reflective symmetry, (c) with four-way rotational symmetry constraints. (d)-(e): Solutions optimized for an as-regular-as-possible (d) or as-fractured-as-possible (e) look by penalizing or prioritizing the appearances of the "butterfly" configurations.

curve. The constraints ensure that the selected half-edges form a closed loop. See Fig. 6 (a) for an example.

4 HYBRID MESHES ON SURFACES

In this section, we begin with the definition and key properties of regular hybrid meshes in 3D. To gain more flexibility, we move on to nearly regular hybrid meshes and show that they are suitable for remeshing a given surface. Section 4.1 introduces *patch graphs*, which are graph structures that facilitate the mapping of multiple 2D regular hybrid meshes onto surfaces to form 3D hybrid meshes. Section 4.2 shows how to generate patch graphs and associated hybrid meshes on a given surface. Finally, in Section 4.3, we introduce a set of editing operations on patch graphs that are useful for pattern exploration.

Regular hybrid meshes in 3D. A *regular hybrid mesh in 3D* is a mesh formed by equilateral triangles and squares. Denoting the sum of angles at the corners of the faces at v by $s(v)$, the *angle defect* $\delta(v)$ equals $360^\circ - s(v)$ for an interior vertex and $180^\circ - s(v)$ for a boundary vertex. A vertex v is called *regular* if it has a zero angle defect, $\delta(v)$, and *irregular* otherwise. In Fig. 8, we show configurations of interior vertices with various angle defects. Obviously, all angle defects are multiples of 30°. This is in contrast to regular triangle meshes and regular quad meshes where angle defects of vertices count in multiples of 60° and 90°, respectively. For an interior vertex v, $\delta(v)$ equals the discrete Gaussian curvature (if angles are measured in radians). Therefore, a regular hybrid mesh in 3D is a discrete surface with vanishing Gaussian curvature except at the irregular interior vertices where Gaussian curvature is concentrated. A special case is provided by Lobel frames which are formed by equilateral triangles only. While angle defects for Lobel frames are just multiples of 60°, the apparent gain of flexibility in terms of angle defects when using regular hybrid meshes is lost on another side, namely the reduced degrees of freedom at vertex stars which include squares. As 3D regular hybrid meshes are an interesting topic, e.g. for architecture, we provide selected examples (see Figures 14 and 15) but focus on the more general 3D hybrid meshes described next.

3D hybrid meshes. Due to the shape limitation of regular hybrid meshes in 3D, we concentrate on *nearly regular* 3D hybrid meshes and simply call them *hybrid meshes*. Our main goal is to provide an algorithm to approximate a given 3D surface by hybrid meshes and to explore the variety of possible solutions.

With any hybrid mesh, we associate *combinatorial quantities* which match the geometric ones in the regular case. We define the *combinatorial angles (c-angles)* at the vertices of a triangle or quad to be 60° or 90°, respectively. Regularity of vertices at hybrid meshes shall be computed based on c-angles. Moreover, we use combinatorial edge lengths (c-lengths) equal to 2 in addition to the geometric ones. Finally, it turns out to be useful to subdivide a hybrid mesh via edge midpoint insertion as in the regular 2D case and this gives rise to c-lengths $\sqrt{3}$ and $\sqrt{5}$. Subdivision splits vertex angles of triangles and quads into 2 or 3 parts, respectively. Their

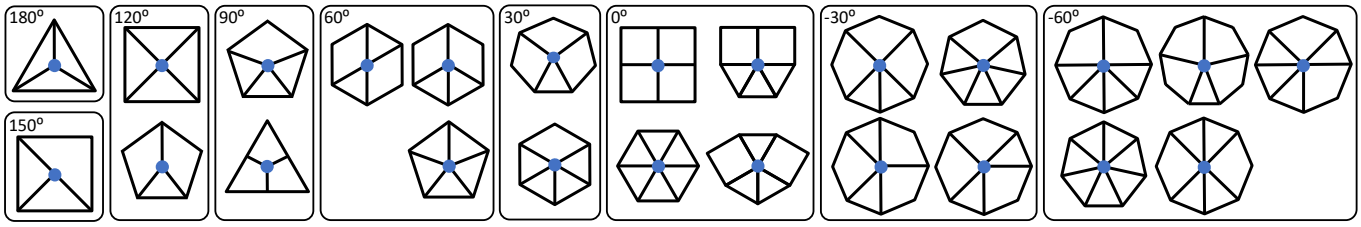


Fig. 8. An exhaustive list of configurations of interior vertices in 3D with angle defects ranging from 180° (the largest possible value) to -60°.

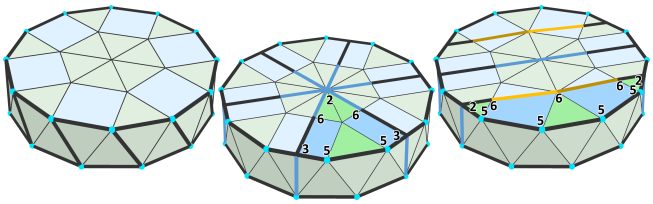


Fig. 9. Multiple ways to construct a patch graph (thick edges) on a 3D regular hybrid mesh derived from a regular 12-sided antiprism. The numbers show the c-indices for the highlighted patches.

c-angle is defined to be 30°, although this is not geometrically true for quads in the regular case.

4.1 Patch graph

Given the subdivided version (Section 3.1), M_s, of a hybrid mesh M in 3D, we define a *patch* as follows. It is a simply connected part of M bounded by a sequence of half-edges in M_s. Further, it shall not contain irregular vertices of M in its interior. For every vertex and edge mid-point along the boundary of a patch, we define its *c-index* as the sum of c-angles inside the patch divided by 30°. See Fig. 3 (c) for an example.

All patch boundary points with a c-index ≠ 6 are called *patch graph vertices*; they split the patch boundary into *patch graph edges* that are formed by sequences of boundary points with c-index 6. Each patch is combinatorially equivalent to a patch in a 2D regular hybrid mesh on which the patch graph edges are main paths. This is an important property which will be used later to combine 2D solutions to 3D hybrid meshes.

A 3D hybrid mesh is always the union of non-overlapping patches. For every partition into patches, the set of patch boundary half-edges plus the c-indices form a *patch graph* of the mesh. See Fig. 9 for examples. In the next section, we describe a patch graph-based strategy for re-meshing a given surface with 3D hybrid meshes.

4.2 Generating hybrid meshes on surfaces

We now present our framework for generating hybrid meshes on surfaces (see Fig. 10 for an overview). The key idea is to partition the surface into sub-surfaces which are suitable as patches of a patch graph. We can then compute a 2D regular hybrid mesh for each of the patches (by the algorithm described in Section 3.3 and its extensions described later) and map them back to the surface by least squares conformal maps. The 2D solutions for the patches are independent of each other if the patch graph edges contain only combinatorial 2-edges. Otherwise we have to care about consistency constraints along graph edges and the process becomes global.

Patch graph construction. Given an input surface S (as a fine polygonal mesh), we would like to re-mesh it by a hybrid mesh M which is close to a regular one. Thus, we have to ensure that combinatorial and geometric angles and edge lengths of M are nearly the same (up to a common scaling factor for the edge lengths). It follows easily that the total Gaussian curvature inside patches has to be close to zero. In other words, we have to nicely distribute the Gaussian curvature at patch graph vertices. This is a guiding principle for patch graph design. We solve it by a *subdivision*-based strategy which can be used in a fully automatic manner, but is also open for user interaction.

The total Gaussian curvature of a mesh can be computed by the discrete Gauss-Bonnet theorem. It states that the total angle defect (i.e., the sum $\sum \delta(v)$ of all angle defects) equals $360^\circ \chi$ where χ is the Euler characteristic of the mesh. Here, one may use combinatorial or geometric angles. The total angle defect can be decomposed into the *total interior angle defect* $TID := \sum \delta(v)$ over all interior vertices v and the sum $B := \sum \delta(v)$ over all boundary vertices v. If angles are computed in a geometric way and faces are planar, then TID is a discrete version of the total Gaussian curvature and B is the total intrinsic turning angle (approximating total geodesic curvature) of the boundary.

A patch of a regular 3D hybrid mesh has $\chi = 1$ and $TID = 0$ since $\delta(v) = 0$ for all interior vertices v. Hence, we have to decompose S into patches for which

$$TID = 360^\circ - \sum_{v \in V_{boundary}} \delta(v) \tag{10}$$

equals zero for c-angles which are close to geometric angles.

To compute TID for a disk-like region R of the input surface, bounded by a polyline b, we first compute B using geometric angles on the 3D surface and then round to the closest integer multiple of 30°. The assignment of c-indices requires care, since we do not yet have the hybrid mesh and we want to compute one which is as regular as possible. For that, let us call boundary points with a geometric angle deficit that is not in $[-15, 15]$ *corners* and the parts between two consecutive corners *sides*. We now assign c-indices I_j to the corners v_j so that (i) they are as close as possible to their inner angles, divided by 30°, and (ii) the combinatorial TID of R equals $360^\circ - \sum_j (6 - I_j)30^\circ$. Our user interface also leaves the user the option to satisfy this equation by introducing c-indices different from 6 to selected vertices at sides. This turns them later into patch

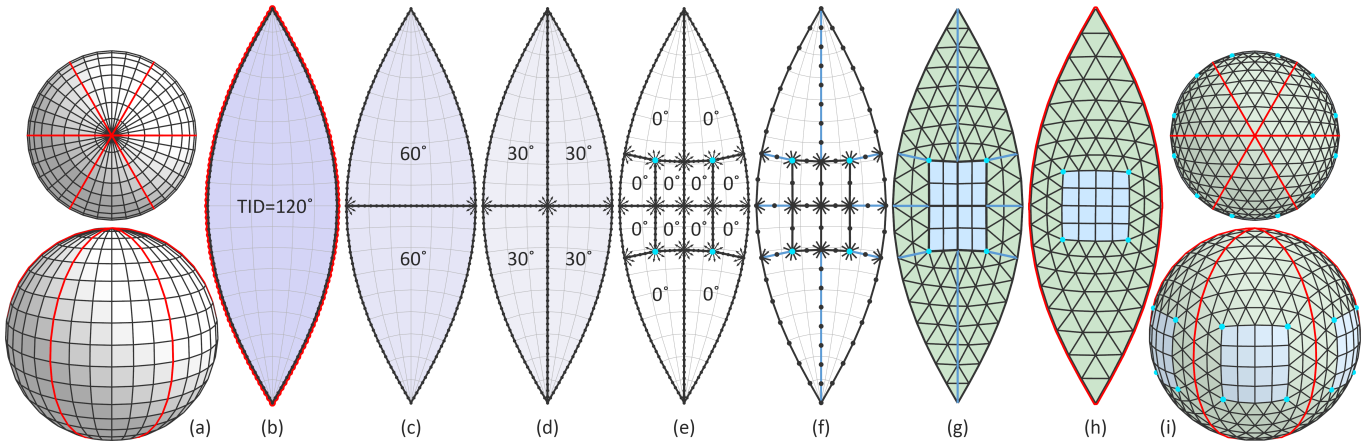


Fig. 10. Overview of our framework. (a) A sphere given as the input surface, which is subdivided into six identical slices. (b)-(e) Subdivision process for one slice. The slice has a total interior angle defect (TID) of 120°. At the end of the construction, there are twelve sub-faces with a zero TID. The four irregular vertices, with an angle defect of 30°, are shown in cyan. (f) The edges are discretized into 2-edges (black) and $\sqrt{3}$-edges (blue). (g) A regular hybrid mesh solution for the whole slice. (h) After geometric post-processing. (i) A final hybrid mesh solution.

graph vertices. Both ways achieve the goal of distributing the total geodesic curvature of the boundary into selected points.

Subdivision process. If the input surface S is not a topological disk, we subdivide it into a collection of sub-surfaces of disk topology. Next, for each sub-surface, we generate a conformal uv parametrization (using LSCM [Lévy et al. 2002] implemented by libigl [Jacobson et al. 2016]). This has the advantage that angles at corners can be computed in the uv domain and that further splitting into patches is simplified. We recursively subdivide the sub-surfaces until all of them have $TID = 0$ as outlined by Algorithm 1.

Algorithm 1 Sub-surfaces subdivision

```
S, a queue of sub-surfaces
S ← the collection of sub-surfaces with a disk-like topology
while S is not empty do
  s ← dequeue from front of S
  TID ← total angle defect of s
  if TID ≠ 0 then
    P ← enumerate all admissible partitions of s
    for all p ∈ P do
      Score p
    end for
    p̂ ← the highest scored p ∈ P
    Partition s by p̂, enqueue new sub-surfaces at end of S
  end if
end while
```

Partitioning a sub-surface S_0 into smaller parts happens in two ways: It can be a binary partition (connecting a pair of boundary vertices on two sides, splitting the sub-surface into two) or an n-ary partition (n being the number of sides of the sub-surface) which splits S_0 into n parts by connecting an interior point of S_0 with one boundary vertex on every side. Splitting shall introduce angles which are close to multiples of 30°. Hence, at a vertex on S_0's boundary with c-index i we have only $i - 1$ possible directions at which the splitting curve has to meet. As splitting curves connecting two boundary vertices we use cubic curves in the uv-domain. A partition is *admissible* if all new sub-surfaces have an absolute value of TID not greater than the absolute value of the TID of S_0. To *score* a partition, we use a heuristic summing up the rounding errors and imposing a large penalty for partitions that run across sharp features of the surface. See Fig. 11 for illustrations.

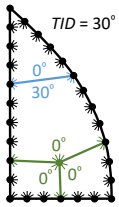


Fig. 11. Partitions of a sub-surface with a total interior angle defect (TID) of 30°. Blue: an admissible binary partition that leads to two parts with TID 0° and 30°. Green: an admissible 3-ary partition that leads to three parts all with $TID = 0°$.

Algorithm 1 can be executed automatically or in a user-guided manner, where at each iteration the user picks one admissible partition among a pool enumerated by the system. We developed a user-interface for the user-guided subdivision process (see the accompanying video).

Turning the final partition into a patch graph. At the end of the subdivision process, we have a graph structure consisting of faces all with a zero TID and the sides of the faces form the edges in the graph. This is not yet a patch graph as we do not yet have a decomposition of the graph edges into sequences of combinatorial 1-edges, 2-edges, $\sqrt{3}$-edges, and $\sqrt{5}$-edges so that each face boundary loop has a geometric realization with the assigned c-angles and c-lengths as Euclidean quantities in the plane. For a more compact description, we just present the case where we have only combinatorial 2-edges.

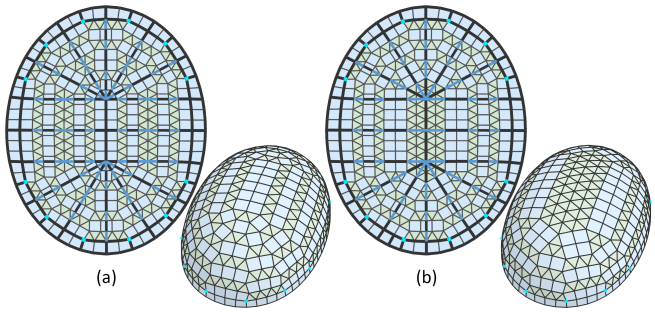


Fig. 12. Two different 3D hybrid mesh solutions for a dome model. They have the same patch graph (thick edges), but differ in the sequential orders of 2-edges and $\sqrt{3}$-edges along the graph edges.

To obtain a patch graph we solve the following IP optimization problem:

$$\begin{aligned} &\text{find} && e_i, 0 \le i < N^e, \text{ such that} \\ &\text{min.} && \sum_i (L_i - 2e_i)^2, \\ &\text{s.t.} && \sum_{0 \le a < 12} \Big(\sum_{i \in I^{a,f_j}} E^a e_i \Big) = 0, \forall\ f_j, \end{aligned} \tag{11}$$

Here N^e is the number of edges in the graph, e_i, $0 \le i < N^e$, are positive integers indicating the number of 2-edges constituting the i-th edge in the graph. L_i, $0 \le i < N^e$, are the actual lengths of the graph edges in $3D$. f_j denotes the j-th face in the graph. a, $0 \le a < 12$, indicates one of the twelve directions. E^a is the six-integer vector of the 2-edge in the a-th direction. I^{a,f_j} indicates the lists of indices of 2-edges circulating f_j in the a-th direction (one edge of every face is arbitrarily chosen to be in the 0-th direction). In short, the objective function is to minimize the sum of squared errors of the differences between the combinatorial edge lengths $2e_i$ and their actual lengths L_i. The constraints are to ensure that for every face, the boundary loop forms a closed loop in the six-integer discrete coordinate system.

After solving Problem (11), the graph structure becomes equivalent to a patch graph (the c-indices are directly derived from the c-angles). The process can be reviewed in Fig. 10 (b)-(f).

3D hybrid mesh generation. For every patch in the patch graph, we generate a 2D regular hybrid mesh (multiple solutions may exist) and then map the hybrid mesh back to the surface using a least squares conformal parameterization. Afterwards, the hybrid meshes are combined together to form a watertight 3D hybrid mesh for the input surface. This process is shown in Fig. 10 (g)-(i).

More interesting patterns can be obtained by discretizing the graph edges into 2-edges, 1-edges, and $\sqrt{3}$-edges (we exclude $\sqrt{5}$-edges for simplicity). This means that Problem (11) is extended to find the sequences of 2-edges, 1-edges, and $\sqrt{3}$-edges for the graph edges. More details of the extended formulation is presented in the additional materials. In addition, cross-patch consistency constraints, described next, are needed to ensure that the per patch solutions join in a seamless manner.

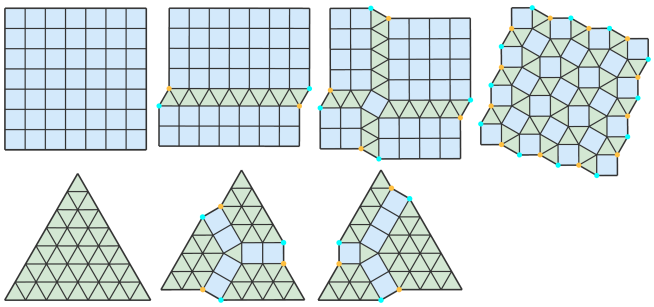
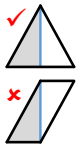


Fig. 13. Editing operations for patches. Top row: the regular polygonal pattern in a 4-sided patch (left) is progressively made more fractured by introducing pairs of zigzags (a vertex with a c-index of 7, orange, followed by a vertex with a c-index of 5, cyan) on its boundary. Bottom row: two ways to introduce a triple of zigzags to the 3-sided patch.

Cross-patch consistency constraints. The first type of constraints ensure that the *sequential order* of different edge types (e.g. 2-edges, 1-edges, and $\sqrt{3}$-edges) along every interior graph edge are consistent for the two adjacent patches. One simple method is to enumerate all or a subset of permutations explicitly. Another more complex formulation can defer the selection of what permutation to choose to the solver. For both adjacent patches, the enumeration of boundary edge placements needs to include all possible permutations of different edge types along that graph edge. Afterwards, additional constraints are added to the IP formulation (Equation 8) to ensure the order of edge types is consistent on both sides. See Fig. 12.

The second type of constraints ensure that partial faces are consistently completed on both sides. For example, a $\sqrt{3}$-edge can be completed in two correct ways to be the mid-line in a triangle and two incorrect ways to be the diagonal of a parallelogram. Constraints need to be added to exclude incorrect completions (see inset).



Post-processing. We use three different numerical optimization algorithms to post-process the geometry of hybrid meshes. The first algorithm optimizes the rotational symmetry of faces using the Euclidean Regularity (ER) term proposed in [Vaxman et al. 2017]. The second algorithm optimizes the global regularity by enforcing globally constant values for edge lengths and diagonal lengths, respectively. The third algorithm optimizes for the planarity of quads, the proximity to a reference surface, and relative fairness (implemented by minimizing the difference between discrete curvatures of original and optimized mesh).

4.3 Patch graph editing

To explore alternative polygonal patterns of 3D hybrid meshes generated by our framework, we propose a set of editing operations that work by altering the *c-indices* of an existing patch graph. The basic idea is to introduce "zigzags" (i.e., a vertex with a c-index of 5 followed by a vertex with a c-index of 7 and vice versa) on the patch boundaries. Our user interface includes an interesting subset of all possible such c-index-editing operations (a detailed explanation is in the additional materials). See Fig. 13 for illustrations.

5 RESULTS AND APPLICATIONS

Table 2 lists the statistics and computation times (on a system with Intel Xeon 16 Core 2.30GHz CPU, 128GB RAM, and NVidia GeForce GTX 1080) for the results shown in the paper. We use Gurobi [Gurobi 2016] to solve the IP problems and Google Ceres Solver [Agarwal et al. 2016] for numerical optimization. For exhaustive enumerations, we observe that the computational cost for retrieving each new solution is roughly linearly proportional to the number of existing solutions to avoid (that we already found previously). Therefore, it may be very expensive to retrieve all the solutions for larger problems such as Fig. 6 and 7 (we successfully did so for Fig. 6 retrieving all the 338 solutions in roughly 6 hrs).

Model	|Input|	|Result|	F_{PG}	E_{PG}	T_{PG}	T_{IP}	T_{post}
Fig. 6 (b)	112e	156T 66Q	1	36	0.76	4.56	1.32
Fig. 6 (e)	112e	156T 66Q	1	36	0.76	4.89	1.27
Fig. 6 (f)	112e	156T 66Q	1	36	0.76	4.88	1.25
Fig. 7 (c)	78e	250T 125Q	1	60	1.01	17.01	3.30
Fig. 7 (e)	78e	250T 125Q	1	60	1.01	11.99	3.41
Fig. 10 pie	108f	152T 16Q	12	28	0.23	0.57	0.55
Fig. 12	3072f	288T 204Q	44	102	0.78	4.87	6.40
Fig. 14 b3D	38f	144T 42Q	38	108	0.42	0.63	0.12
Fig. 14 tatT	62f	380T 120Q	62	180	1.2	1.02	1.91
Fig. 16	576f	984T 492Q	79	306	3.36	6.57	143.94
Fig. 18 (a)	5400f	271T 532Q	12	37	1.02	3.42	48.04
Fig. 18 (b)	5400f	788T 346Q	16	129	1.33	4.09	60.65
Fig. 18 (c)	5400f	770T 365Q	25	345	6.66	5.56	64.83
Fig. 19 (a)	3312f	668T 380Q	16	80	1.88	4.68	47.95
Fig. 19 (b)	3312f	756T 380Q	38	146	1.92	4.73	57.33
Fig. 19 (c)	3312f	886T 405Q	36	204	2.07	5.74	76.61
Fig. 20 L	8526f	1005T 586Q	30	84	1.25	9.01	136.63
Fig. 20 R	8626f	1029T 619Q	32	202	3.88†	7.87	158.82
Fig. 21 (a)	5367f	0T 1496Q	22	63	1.47	7.25	195.59
Fig. 21 (b)	5367f	1006T 950Q	30	97	1.52	7.90	173.82
Fig. 21 (c)	5367f	1070T 918Q	26	93	1.71	9.49	176.95

† 2-edges only.

Table 2. Statistics. |Input| is the face count of the 3D input surface or the edge count of the 2D input boundary loop. |Result| is the face count (triangles and quads) of the resulting hybrid mesh. F_{PG} and E_{PG} are numbers of patches and edges (i.e., sides) in the patch graph. T_{PG}, T_{IP}, and T_{post} are time (in seconds) for solving Problem 11 (patch graph generation), Problem 8 (hybrid meshing using IP), and geometric post-processing, respectively.

3D Hybrid Mesh Design. Using our framework, we propose several strategies for 3D hybrid mesh design. In Fig. 14, we explore designs from polyhedra. In Fig. 15, we generate 3D designs by folding a 2D regular hybrid mesh. Finally, in Fig. 16, we design a lamp model with a hybrid mesh as an artistic decorative pattern. To see how our design works with lighting, we printed a 3D model shown in Fig. 17.

Architectural Geometry. To illustrate the power of our hybrid meshing technique, we have created novel triangle-quad patterns for three well-known architectural freeform surfaces (see Figures 18, 19, 20, and 21). Such patterns have not been seen in architecture before. An important criterion in this application is the planarity of panels. We could confirm the conjecture that the probability for success of an optimization towards planarity of quads is high for fractured patterns (see Fig. 22). Clusters of quads, if not aligned correctly with the curvature behavior of the surface, lead to a failure of planarization or severe distortion of the pattern. For details on underlying geometric facts, see e.g. [Pottmann et al. 2015].

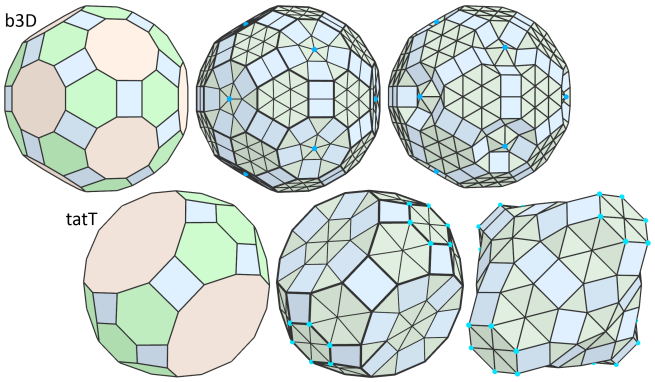


Fig. 14. 3D hybrid mesh design from polyhedra. We first convert polyhedra with Conway polyhedral notations "b3D" (top row) and "tatT" (bottom row) to patch graphs. The "b3D" polyhedron has twelve 10-sided and the "tatT" polyhedron has four 12-sided faces. In the middle, we show 3D hybrid meshes generated from patch graphs directly derived from the polyhedra. On the right, the hybrid meshes are made as-regular-as-possible by the ER-based geometry optimization (the "b3D" result achieved full regularity).

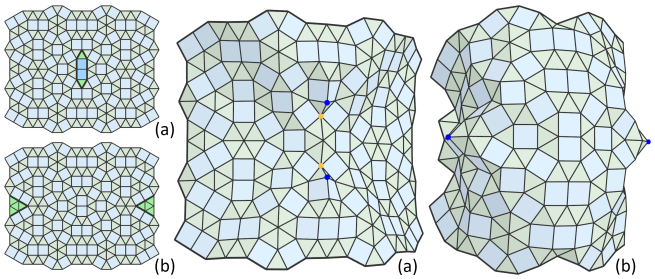


Fig. 15. We can turn 2D regular hybrid meshes into interesting 3D shapes. We change the combinatorics by cutting out parts of the mesh and gluing the newly created boundaries together. Then, we use numerical optimization to compute 3D regular hybrid meshes with the new combinatorics. We reduce the degrees of freedom with a cylindrical reference surface.

6 CONCLUSIONS AND FUTURE WORK

A limitation of our current framework is that we do not have a theoretical study of necessary and sufficient conditions that a given 2D boundary admits at least one regular hybrid mesh solution. At the moment, we can only establish the answer to this question computationally. If the IP solver returns no solutions it means that no solution exists. The simplest necessary condition is related to the area enclosed by the given boundary. A hybrid mesh with m squares and n triangles has area $4m + \sqrt{3}n$. This implies that all possible solutions inside a boundary must have the same number of triangles and the same number of quads. Therefore, the area inside a given boundary gives a simple necessary condition for the existence of a solution (See Fig. 23 left for an example of a boundary that admits no solution due to the area condition).

However, even if the simple area test passes, there are boundaries that admit no solutions. Given an arbitrary boundary, the likelihood that problem 8 is infeasible is considerable. Often, these boundaries

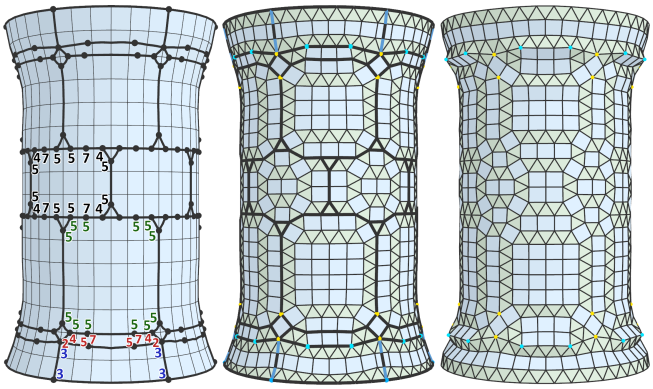


Fig. 16. Making the lamp model. Left: the patch graph. The c-indices are shown for vertices of selected patches (one color for one patch). Here we utilized the patch editing operations (Section 4.3) for pattern design. A key strategy is to assign c-indices 5 and 7 on to the same vertex in two adjacent patches to keep it regular. Middle: 3D hybrid mesh result (the patch graph is shown in thicker edges). Right: After geometric post-processing.

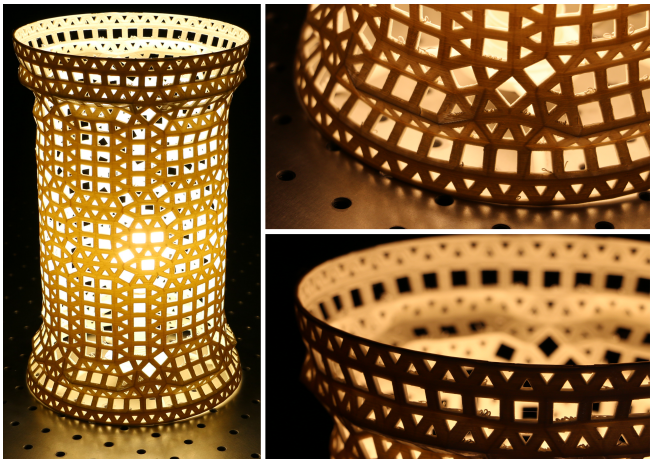


Fig. 17. 3D printed lamp model.

can be meshed by adding an additional shape to the tile set, a rhombus with a 30° at its sharpest corner (that has an area of 2). We implemented this extended meshing algorithm and show a result in Fig. 23 middle. We can also setup the computation to move the rhombi towards the boundary to compute the minimal change to the boundary that would enable a solution. (See Fig. 23 right). We propose the theoretical investigation of the existence of solutions as interesting avenue for future work.

As a consequence of the aforementioned limitation, a bottleneck of our framework is that the prescribed patch boundary division and combinatorial angles after a patch graph is constructed may admit no feasible solutions for some of the patches. Therefore, considerable effort is needed to ensure the feasibility of 2D solutions when manually designing patch graphs. We currently workaround this problem by having a patch graph construction strategy that starts from one with known 2D solutions (such as a patch graph with only 3-sided and 4-sided patches) and create variations by editing operations that we know would preserve the feasibility of 2D solutions (such as the ones described in Sec. 4.3). However, this probably limited the possibility of designs being explored. We believe that there is a lot of room for improving the patch graph construction in future work.

Further, we would like to look at volumetric extensions. Here, one can observe that it is not possible to mesh with tets and cubes alone, because their faces do not match. Therefore, it would be interesting to understand what (volumetric) tile sets are feasible while at the same time providing meshes of practical relevance.

Conclusions. In this paper, we introduced the concept of triangle-quad hybrid meshes. While special forms of these meshes, such as quad dominant meshes, were used before, this is the first extensive computational treatment of this topic. We proposed a first framework for designing triangle-quad hybrid meshes on surfaces and show the computational design of a new type of meshes that only could be generated manually before. The core algorithmic contribution is an IP formulation that can exhaustively enumerate hybrid meshes inside a given boundary.

Acknowledgments. This research was supported by the KAUST Office of Sponsored Research and the Visual Computing Center (VCC). This research was also supported by SFB-Transregio programme Geometry and Discretization (Austrian Science Fund grant no. I 2978). We thank Yu Tian for the 3D printing and Qiang Fu for the photography of the lamp model. We thank Heinz Schmiedhofer for the 3D rendering and Evolute GmbH for the planarity optimization. We thank Eugene Zhang, Yue Zhang, Chengcheng Tang, Olga Diamanti, and Caigui Jiang for invaluable discussions.

REFERENCES

Sameer Agarwal, Keir Mierle, and Others. 2016. Ceres Solver. http://ceres-solver.org. (2016).

Abdalla G. M. Ahmed, Hélène Perrier, David Coeurjolly, Victor Ostromoukhov, Jianwei Guo, Dong-Ming Yan, Hui Huang, and Oliver Deussen. 2016. Low-discrepancy Blue Noise Sampling. *ACM Trans. Graph.* 35, 6, Article 247 (Nov. 2016), 13 pages.

Pierre Alliez, Éric Colin de Verdière, Olivier Devillers, and Martin Isenburg. 2005. Centroidal Voronoi Diagrams for Isotropic Surface Remeshing. *Graph. Models* 67, 3 (May 2005), 204–231.

David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, and Denis Zorin. 2013. Quad-Mesh Generation and Processing: A Survey. *Comput. Graph. Forum* 32, 6 (Sept. 2013), 51–76.

Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Levy. 2010. *Polygon Mesh Processing*. A K Peters/CRC Press.

Weikai Chen, Yuexin Ma, Sylvain Lefebvre, Shiqing Xin, Jonàs Martínez, and Wenping Wang. 2017. Fabricable Tile Decors. *ACM Trans. Graph.* 36, 6, Article 175 (Nov. 2017), 15 pages.

Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. 2003. Wang Tiles for Image and Texture Generation. *ACM Trans. Graph.* 22, 3 (July 2003), 287–294.

David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. 2004. Variational Shape Approximation. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 905–914.

Branko Grunbaum and Geoffrey Colin Shephard. 2016. *Tilings and Patterns: Second Edition*. Dover Publications.

Gurobi. 2016. Gurobi Optimizer Reference Manual. (2016). http://www.gurobi.com

Alejo Hausner. 2001. Simulating Decorative Mosaics. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. 573–580.

Joachim Hermisson, Christoph Richard, and Michael Baake. 2002. A Guide to the Symmetry Structure of Quasiperiodic Tiling Classes. 7 (05 2002).

Wenchao Hu, Zhonggui Chen, Hao Pan, Yizhou Yu, Eitan Grinspun, and Wenping Wang. 2016. Surface Mosaic Synthesis with Irregular Tiles. *IEEE Transactions on Visualization and Computer Graphics* 22, 3 (March 2016), 1302–1313.

Alec Jacobson, Daniele Panozzo, et al. 2016. libigl: A simple C++ geometry processing library. (2016). http://libigl.github.io/libigl/.

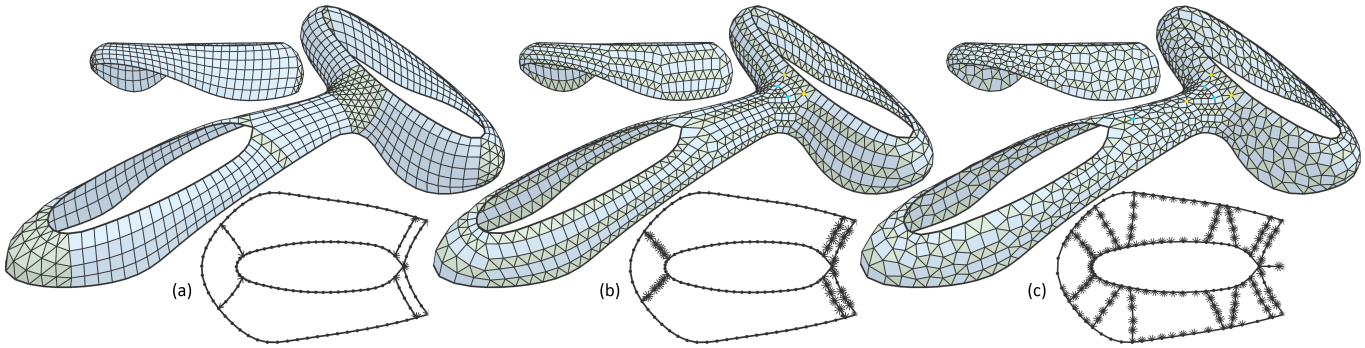


Fig. 18. 3D hybrid mesh designs for the freeform surface at the Yas Island Hotel, Abu Dhabi. (a) A baseline design that can be seen as the combination of multiple triangle and quad meshes. (b) Horizontal stripe patterns are created by adding "zigzags" along the patch graph edges in the vertical direction. (c) Fractured patterns are created by interleaving stripes in both horizontal and vertical directions. In each result, we show two views of the model and a part of the patch graph (near the front end of the model). We illustrate the c-indices $\neq 6$ of vertices with little spikes pointing into the corresponding patches.

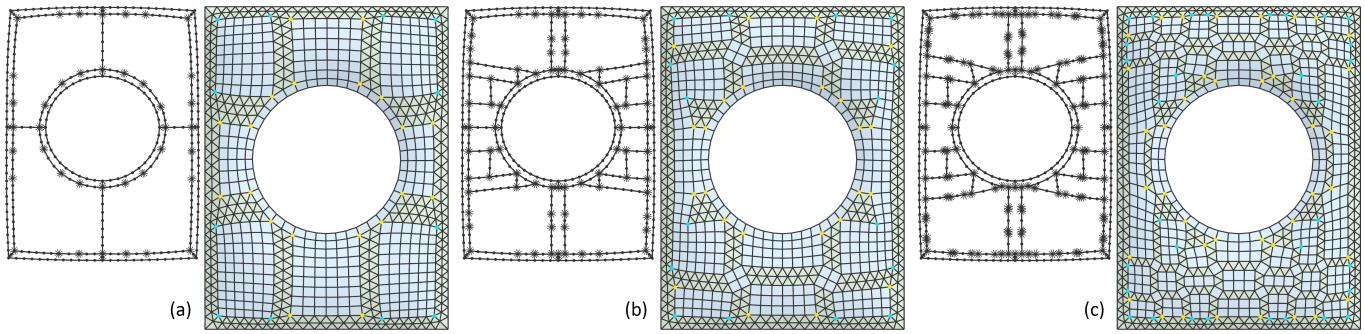


Fig. 19. Three 3D hybrid mesh designs for the Great Court of the British Museum with increasingly complex patterns. We show one selected solution for each patch graph. Note that a patch graph can have many solutions, e.g., patch graph (c) admits 81 hybrid mesh solutions. Some are shown in the additional materials.

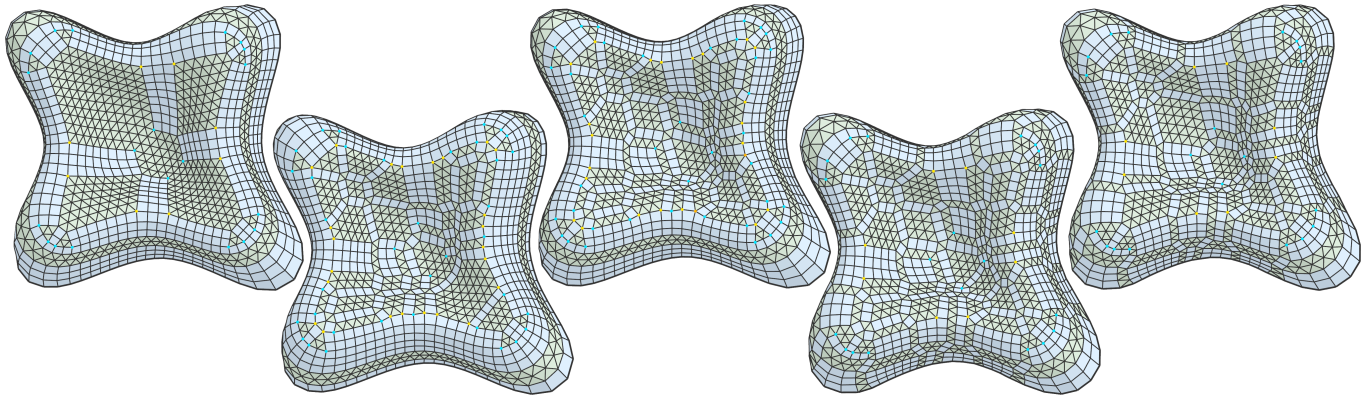


Fig. 20. 3D hybrid mesh designs for the model of the Lilium tower by Zaha Hadid Architects.

Caigui Jiang, Chengcheng Tang, Amir Vaxman, Peter Wonka, and Helmut Pottmann. 2015. Polyhedral Patterns. *ACM Trans. Graph.* 34, 6, Article 172 (Oct. 2015), 12 pages.

Craig S. Kaplan and David H. Salesin. 2000. Escherization. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. 499–510.

Craig S. Kaplan and David H. Salesin. 2004. Dihedral Escherization. In *Proceedings of Graphics Interface 2004 (GI '04)*. 255–262.

Junhwan Kim and Fabio Pellacini. 2002. Jigsaw Image Mosaics. *ACM Trans. Graph.* 21, 3 (July 2002), 657–664.

Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. 2006. Recursive Wang Tiles for Real-time Blue Noise. *ACM Trans. Graph.* 25, 3 (July 2006), 509–518.

Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. 2002. Least Squares Conformal Maps for Automatic Texture Atlas Generation. *ACM Trans. Graph.* 21, 3 (July 2002), 362–371.

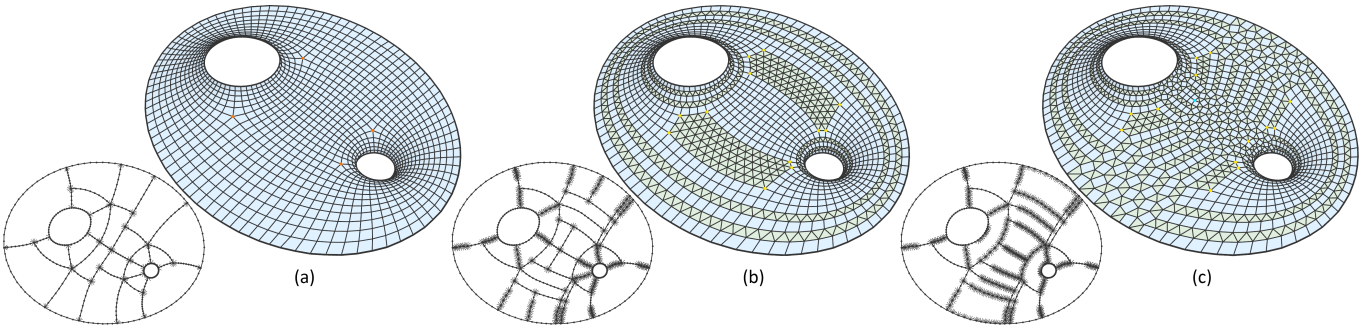


Fig. 21. Three hybrid mesh designs for a freeform architectural surface. We begin with a pure quad mesh design (a) and gradually introduce fractured patterns to produce (b) and (c). In (b), we also break each of the four irregular vertices with an angle defect of 90° into three irregular vertices with an angle defect of 30°. The patch graphs are shown in the bottom-left corners.

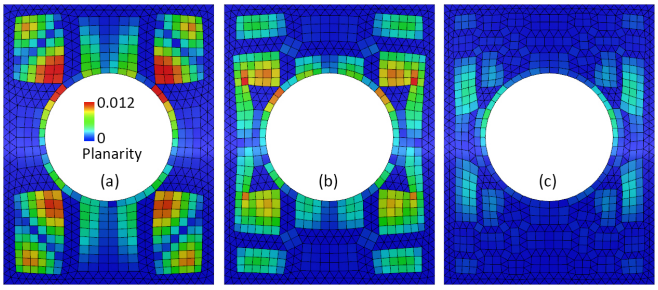


Fig. 22. Meshes (a), (b) and (c) from Fig. 19 after optimization for planarity with the same tolerance for proximity to the reference surface. The planarity value per quad is the diagonal distance, divided by the mean diagonal length. Only the most fractured model (c) is below the practical tolerance of 0.005 for this value.

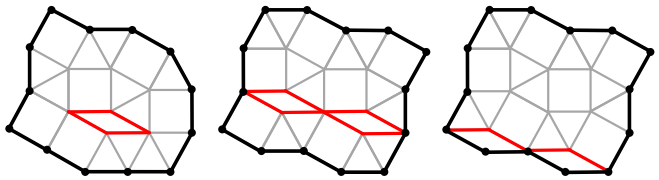


Fig. 23. Left: A 2D boundary with uniform-length edges and angles in multiples of 30° that admits no regular hybrid mesh solution. It also failed the the area condition (since it can be meshed with the addition of a rhombus). Middle: A boundary that passes the area condition but still admits no solution. Interestingly, it can be meshed with the addition of two rhombi. Right: Computing a solution with rhombi on the boundary suggests a minimal change to the boundary that enables a solution.

Yufei Li, Yang Liu, and Wenping Wang. 2015. Planar Hexagonal Meshing for Architecture. 21 (09 2015), 95–106.
Alain Lobel. 2004. Lobel Frames: Forms and structures generated by identical elements. (2004). http://www.equilatere.net
Giorgio Marcias, Kenshi Takayama, Nico Pietroni, Daniele Panozzo, Olga Sorkine-Hornung, Enrico Puppo, and Paolo Cignoni. 2015. Data-driven Interactive Quadrangulation. *ACM Trans. Graph.* 34, 4, Article 65 (July 2015), 10 pages.
Chi-Han Peng, Yong-Liang Yang, and Peter Wonka. 2014. Computing Layouts with Deformable Templates. *ACM Trans. Graph.* 33, 4, Article 99 (July 2014), 11 pages.
Helmut Pottmann, Michael Eigensatz, Amir Vaxman, and Johannes Wallner. 2015. Architectural Geometry. *Computers and Graphics* 47 (2015), 145–164.
Bernhard Reinert, Tobias Ritschel, and Hans-Peter Seidel. 2013. Interactive By-example Design of Artistic Packing Layouts. *ACM Trans. Graph.* 32, 6, Article 218 (Nov. 2013), 7 pages.
Pedro V. Sander, John Snyder, Steven J. Gortler, and Hugues Hoppe. 2001. Texture Mapping Progressive Meshes. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. 409–416.
Scott Schaefer and Joe Warren. 2005. On C2 Triangle/Quad Subdivision. *ACM Trans. Graph.* 24, 1 (Jan. 2005), 28–36.
Christian Schumacher, Bernhard Thomaszewski, and Markus Gross. 2016. Stenciling: Designing Structurally-Sound Surfaces with Decorative Patterns. *Comput. Graph. Forum* 35, 5 (Aug. 2016), 101–110.
Jos Stam and Charles Loop. 2003. Quad/Triangle Subdivision. *Computer Graphics Forum* 22, 1 (2003), 79–85.
Jens-Boie Suck, Michael Schreiber, and Peter Häussler. 2002. *Quasicrystals. An Introduction to Structure, Physical Properties and Applications.* Springer-Verlag.
Kenshi Takayama, Daniele Panozzo, and Olga Sorkine-Hornung. 2014. Pattern-Based Quadrangulation for N-Sided Patches. *Comput. Graph. Forum* 33, 5 (Aug. 2014), 177–184.
Amir Vaxman, Christian Müller, and Ofir Weber. 2017. Regular Meshes from Polygonal Patterns. *ACM Trans. Graph.* 36, 4, Article 113 (July 2017), 15 pages.
Shi-Qing Xin, Bruno Lévy, Zhonggui Chen, Lei Chu, Yaohui Yu, Changhe Tu, and Wenping Wang. 2016. Centroidal Power Diagrams with Capacity Constraints: Computation, Applications, and Extension. *ACM Trans. Graph.* 35, 6, Article 244 (Nov. 2016), 12 pages.
Yong-Liang Yang, Jun Wang, Etienne Vouga, and Peter Wonka. 2013. Urban Pattern: Layout Design by Hierarchical Domain Splitting. *ACM Trans. Graph.* 32, 6, Article 181 (Nov. 2013), 12 pages.