Tile patterning on free-form surfaces that reduces tile cutting

Chaoyu DU*, Tom VAN MELE, Philippe BLOCK

*Block Research Group, ETH Zurich
Stefano-Franscini-Platz 1, 8093 Zurich, Switzerland
chaoyu.du@arch.ethz.ch

Abstract

We propose a new strategy for generating staggered patterns for standard, rectangular tiles on free-form surface. The given input surface is firstly tessellated into near-square quadrilateral faces using a re-meshing algorithm. Afterwards, we run a matching algorithm that assigns each tile to two adjacent faces. We use binary integer programming to find a tessellation on the surface that minimises the number of unique tiles and tiles that are not staggered. Lastly, we generate the tiles and post-process the tessellation result.

Keywords:

tile vaulting, tile pattern, free-form surface, quadrangulation, binary integer programming

1. Introduction

Shells and vaults are efficient structures whose form follows the flow of forces. They emerge again in contemporary architecture because of the emergence of computational form-finding techniques to design them as well as for their sustainability. Specifically, for longer spans, shell structures require less material quantities and for funicular vaults lower-strength materials, which typically have reduced embodied carbon coefficients [1]–[3].

Among the methods of constructing shells and vaults, thin-tile vaulting is a 600-year-old Mediterranean technique that requires minimum formwork during its construction and relies on traditional manufacturing of the tiles. The flexibility offered by non-uniformly spacing of the tiles, by using small offset deviations in the mortar joints, a wide range of shell geometries can be achieved with standard tile shapes. However, tile vaulting dropped in popularity due to (among many factors) the high cost of skilled labour, emerging competitive materials, namely steel and concrete, modern building codes and the onset of modernism that favoured straight lines [4]–[6] . The construction of tile vaults involves skilled masons adjusting the direction of the tile, handling the mortar thickness, and cutting the tiles on site dextrously to follow the curvature and gradually fill the surface. This process can become especially challenging when the surface is geometrically intricate [7], [8].

Recent research suggests that this complexity can be partially overcome with the help of robotic fabrication and augmented reality, as the tiles can be precisely registered in 3D. The craft-specific tasks can be executed in a fully automated manner or by collaboration between humans and machines [9]–[11]. However, the main challenge remains in the generation of tiling patterns, i.e., where, and how to place the single-sized tiles on a free-form surface.

This paper focuses on how to compute a tiling pattern on a free-form surface and tries to find a global solution to the tiling pattern that can achieve both aesthetic requirements and reduce fabrication complexity, which means a minimal amount of customised cutting on-site. Specifically, this paper focuses on the shells and vaults with small to medium span and surface curvature large compared with the tile size.

2. Related work

2.1 Surface geometry and tile pattern

The process of traditional tile laying involves a series of technical decisions that impact various aspects of a structure, such as its strength, stability, aesthetics, constructability, and cost-effectiveness [1], [6], [8]. This section will discuss two examples of tile-laying patterns, examining the challenges and opportunities they present in terms of research and construction.

Guastavino vaulting is not only structural but also decorative [6]. The sophisticated decorative tile patterns exposed in their buildings (Figure 1. left) are accomplished by highly skilled masons by applying a final decorative layer from below. This step allowed the creation of more complex tile patterns without consideration of the construction sequence. However, in some cases, it's simply impossible to use a single pattern to fill a surface, for example, using a herringbone pattern to construct a dome without compensating for the smoothness and customised cutting (Figure 1. middle). In such situations, Guastavino uses multiple pattern patches instead of using a single pattern, which not only makes the vault decorative but also resolves the tiling problem for the three-dimensional forms.

The method of separating the shell in a full or partial manner also exists in other designs, to accommodate the surface aesthetics and tessellation pattern. For example, the glass vault is a doubly-curved barrel vault built by two robotic arms without a centring [10]. It adopts a herringbone pattern to interlock the bricks and defines a clear construction sequence for the robots to build alongside the first arch, and from bottom to top. To accommodate the amplified gap sizes due to curvature changes at the scale of the final structure, the tessellation pattern is shifted by half a brick along the reset lines (Figure 1. right), along which the herringbone pattern stops. Moreover, the vault uses irregular tiles at the boundary to ensure compete pattern on the shell surface.

Figure 1. left: some examples of Guastavino tiling patterns[6]; middle: the dome at the Chapel of Our Lady (photo credits: Michael Freeman); right: tiling pattern and reset lines of the glass vault [10].

2.2 Computational tile and brick laying

A few researchers have tackled the problem of tile laying to find a suitable mediation between aesthetics and structural behaviour. Shaghayegh [12] developed a design tool that can semi-automatically tessellate bricks on a curved surface by sequentially projecting the tiles. The workflow requires the designer to first choose a pattern in 2D and the start point of the first tile. Afterwards, the algorithm can compute the location of the adjacent tile by referring to the previous ones. However, the transformation and projection of the next tile is based on the local positions of the previous tiles, and thus globally, cutting the tiles is inevitable.

Adiels et al. [13] compute the geodesic distance on the curved surface to guide their tessellation process. This approach yields several benefits, such as effectively aligning patterns with boundaries and enabling

the creation of running bond designs. However, this method faces challenges to accommodate other tiling patterns and singularities, where tiles need to be arranged circularly.

Unlike these methods based on geometry, Panozzo et al. [14] formulate the patterning as a 2-colouring graph problem on a quad mesh, i.e. the staggered pattern is created by removing every second edge of a regular grid. Although this research does not focus on tessellation with rectangular tiles, the approach of combining geometric and topological methods serves as an inspiration for our research.

2.3 Surface paneling and rationalisation

Surface panelling, a related topic to tile laying, involves filling a curved surface with geometric or manufacturing constraints. The problem is typically tackled through a two-step process: first, segmenting the shape into smaller pieces, and then, approximating each segment with a panel within a given tolerance [15], [16].

According to Pottmann et al. [17], good segmentation is crucial to achieving an accurate approximation. This can be accomplished through techniques such as polygonal mesh layout, re-meshing, and patch decomposition. Though segmentation is usually a decision driven by the designers, a good segmentation can lead to a good approximation.

One typical approximation task is driven by the need to fabricate the panels with a limited number of moulds to reduce the overall cost. Eigensatz et al. [16] assign the panel type, such as planar, cylindrical or custom, to each segment and interleave discrete and continuous optimisation steps to improve the quality of the panelling and to reduce the number of moulds while still keeping the kink angles and gaps within tolerance.

Approximation can also be driven by a desired panel shape. Shape-up [18] is an optimisation framework for geometry processing that enforces shape constraints in a local-global iterative algorithm. In the local step, the distance between each vertex in the mesh and its projections under multiple constraints is encoded. The global step then minimally relocates the vertices on the mesh to match the projected fragments and solves a single energy function to minimise deviation from the local shape approximation, input surface, and surface smoothness.

3. Proposed approach

In this paper, we propose a three-step method to generate tile-laying patterns on a free-form surface, as shown in Figure 2. This method finds a global solution to tessellate the surface and while keeping the need for unique cutting small.

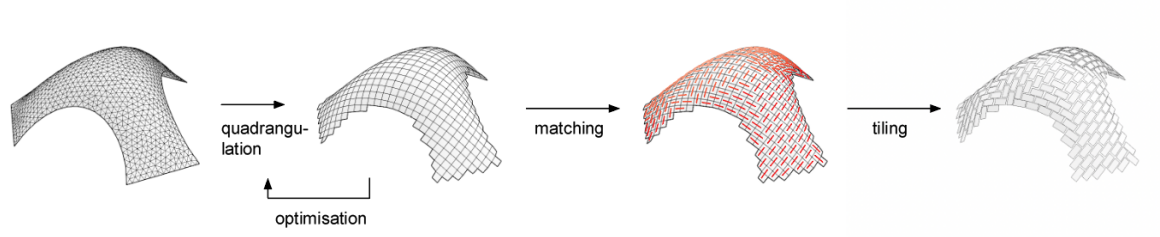


Figure 2. The proposed three-step tile patterning workflow.

The three steps are:

1. quadrangulation: generating a global quad mesh from the input surface;
2. matching: finding a (mostly) surface-covering combination of pairs of two neighbouring quadrilateral faces; and,
3. tiling: generating tile geometries.

3.1. Quadrangulation

We first convert the input surface into a quad mesh. Quadrangulation is an important topic in computer graphics and architectural geometry [19]. Similar to the panelling task described in Section 2.3, a good initial quadrangulation can reduce the efforts of post-processing and further optimising the mesh surfaces. We aim to partition our reference surface into a semi-regular quad mesh, whose quadrilateral faces are close to squares of equal size and where the number of singularities remains small.

We have considered two approaches. The first one is based on parametrisation, which means mapping the surface embedded in 3D to a domain in 2D and tessellating the 2D domain by regular grids. However, to reduce distortion, we need to find a correct set of seams either to cut the geometry and restitch it or the correct placement of singularities, which is challenging to compute and easily leads to areas with locally high distortion. These situations are not ideal for our tile arrangement in the next steps.

Thus, we instead adopt a field-guided approach, which uses an orientation field and prioritises higher distortion reduction. This approach is typically composed of three steps:

1. Generate an orientation field. This orientation field is driven by principal curvature and defined continuously over the surface. If the field is locally not smoothly aligned in a region, a singularity can occur, resulting in an irregular vertex in the quad mesh, which in our case, means an irregular tile.
2. Compute a position field, determining where the vertices are placed. Since square faces are preferred, the sizing function should be isotropic.
3. Synthesise the quad mesh.

We use the QuadriFlow algorithm without boundary constraints for this re-meshing step [20]. It allows for the efficient production of scalable quad meshes with minimal distortion while maintaining a low number of singularities. We use the tile dimension to estimate the target mesh edge length. In general, the algorithm provides a reasonable approximation of the half-tile locations; however, some level of distortion is inevitable. Here we present two quadrangulation outcomes for a surface (as illustrated in Figure 3, left and middle), and evaluate their area and angle distortion with respect to the squares. For this surface, the presence of the singularity can cause the faces near it to become smaller and more distorted, resulting in overlapping tiles. If the singularity is removed, bigger gaps will appear in the middle of the shell. If neither of the two solutions is satisfactory, we can further enhance the mesh quality by imposing square shape constraints. For the latter, we use the shapeup optimisation solver implemented in libigl [21], and the outcome is displayed in Figure 3, right. It is important to note though that due to the shapeup optimisation the shape of the mesh deviates from the original input shape.

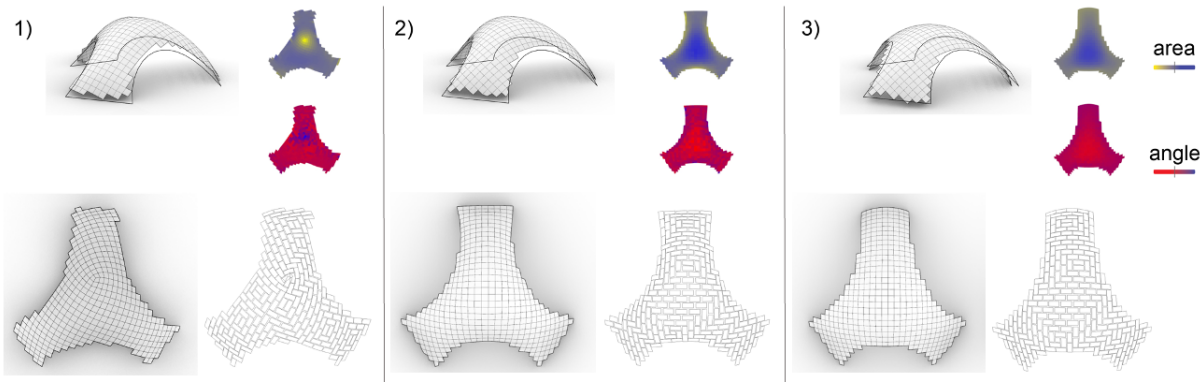


Figure 3. Left and middle: quadrangulation result with singularities without singularities; right: quadrangulation result after shapeup optimisation.

3.2. Maximal matching

Section 3.1 has discussed how to generate a high-quality quadrilateral mesh with faces close to squares of equal size. Based on this mesh, an arrangement of rectangles can be generated by connecting pairs of adjacent faces. One face can only be connected either 1 time or 0 times. If it is not connected, the face represents a half-tile, which needs additional cutting. Thus, maximising the pair of adjacent faces to generate as many tiles as possible can be formulated as a graph-matching problem.

Given the set of faces F and the possible edges E that connect adjacent faces, the goal is to find the maximal number of matched edges E', where E' is a subset of E, such that no two edges in E' share an adjacent face.

This problem can be formulated as a binary integer program:

Let edge connectivity $\mathrm{x_e} \in \{0, 1\}$, $x_e = 1$ if $e \in E'$, else $x_e = 0$. For one face, let $E(f)$ denote all edges that connect a face f. Taking Figure 4. left as an example. For face f_4, the edges that connect to it are listed in $E(f_4) = \{e_{14}, e_{34}, e_{45}, e_{47}\}$. One face can be at most connected to one adjacent face. If f_4 and f_7 are connected, for example, $x_{47} = 1$ and $\mathrm{x}_{14} = \mathrm{x}_{34} = \mathrm{x}_{45} = 0$.

Our problem can be written as follows:

$$\begin{aligned} &\max \sum_{e \in \mathrm{E}} x_e \\ &s.t. \sum_{e \in \mathrm{E(f)}} x_e \leq 1, \forall f \in \mathrm{F} \end{aligned} \tag{1}$$

We use the Gurobi Optimizer [17] to solve the binary integer program (1). Figures. 3-2, middle and right, show the matching result for a 10x10 grid and an 11x11 grid. The total number of faces for an 11x11 grid is odd, resulting in one unmatched face. Note that many edges are parallel, which can cause continuous mortar joints in real tile-laying. We address this problem in the next section.

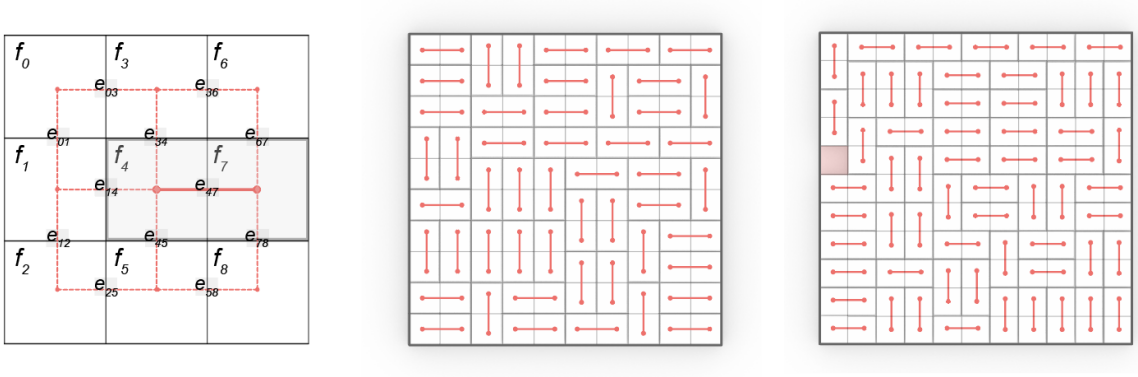


Figure 4. left: A mesh composed of nine faces and the connectivity between them. f_4 and f_7 are a connected face pair; middle and right: the matching result for a 10x10 grid and an 11x11 grid using binary integer program (1).

3.3 Staggered pattern

In tile vaulting, the tiles are often staggered to avoid continuous mortar. To avoid parallel edges, as is the case in Section 3.2, we can formulate them as penalties in the objective function.

We introduce a new set of variables $\mathrm{y_{e1,e2}}$ to describe neighbouring parallel edges. Let $\mathrm{y_{e1,e2}} \in \{0, 1\}$, $\mathrm{y_{e1,e2}} = 1$ if and only if both $e_1, e_2 \in E'$, else $\mathrm{y_{e1,e2}} = 0$. Taking the example in Figure 4. left, edge e_{47} has only two neighbouring parallel edges e_{36}, and e_{58}. We know that f_4 and f_7 are connected, so $x_{47} = 1$.

Assume f_3 and f_6 are connected, $x_{36} = 1$. Now, we have a pair of parallel neighbouring edges, so $y_{e47,e36} = 1$. In our objective, we can penalise this situation.

The optimisation problem with penalties can be written as follows:

$$\begin{aligned} &\max \sum_{e\in E} x_e - w * \sum y_{e1,e2} \\ &s.t. \sum_{e\in E(f)} x_e \leq 1, \forall f \in F, \\ &x_{e1} + x_{e2} - y_{e1,e2} \leq 1, \forall y_{e1,e2} \end{aligned} \tag{2}$$

The weight w for the penalty term can be chosen to tell the optimiser how it should balance between avoiding unmatched faces and parallel edges if no solution is possible that avoids both.

The solutions found by Gurobi for the 10x10 and 11x11 grids are shown in Figure 5. The results show high similarities to state-of-the-art patterns observed in tile vaults built by expert masons. Our algorithm can also generate a staggered pattern for mesh with singularities (Figure.5 right).

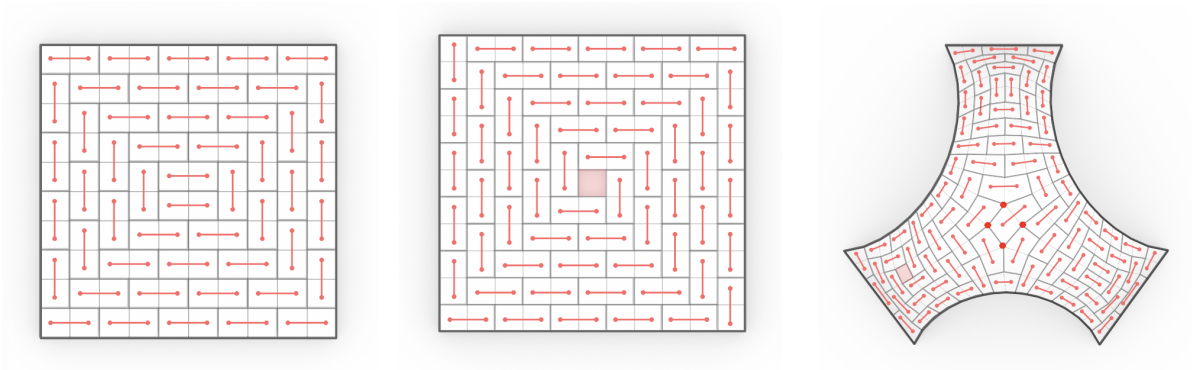


Figure 5. left and middle: staggered pattern for a 10x10 grid and an 11x11 grid, respectively; and right: staggered pattern for mesh with singularities, which are shown as red dots.

3.4 Other patterns

Other tile patterns, e.g., a running-bond or herringbone pattern, can be targeted by replacing the penalty term for parallel edges described in Section 3.3 with a different term that penalises or rewards certain combinations of edges.

For example, we can optimise for a running-bond pattern by using a term that rewards pairs of parallelly shifted edges, as shown in Figure 6. left (i.e. the edges are in neighbouring columns or rows of the grid but "shifted" by one row/column). This can be done by introducing a new set of binary variables $z_{e1,e2} \in \{0, 1\}$, which exists for every neighbouring shifted pair $e_1, e_2 \in E$. Again, as in Section 3.3, we add a set of constraints that enforces $z_{e1,e2} = 1$ if and only if both $e_1, e_2 \in E'$. These constraints look different because the $z_{e1,e2}$ contributes positively to the objective function while the $y_{e1,e2}$ in Section 3.2 contribute negatively. Our problem is written as follows:

$$\begin{aligned} &\max \sum_{e\in E} x_e + w_R * \sum z_{e1,e2} \\ &s.t. \sum_{e\in E(f)} x_e \leq 1, \forall f \in F, \\ &z_{e1,e2} \leq x_{e1}, z_{e1,e2} \leq x_{e2}, \forall z_{e1,e2} \end{aligned} \tag{3}$$

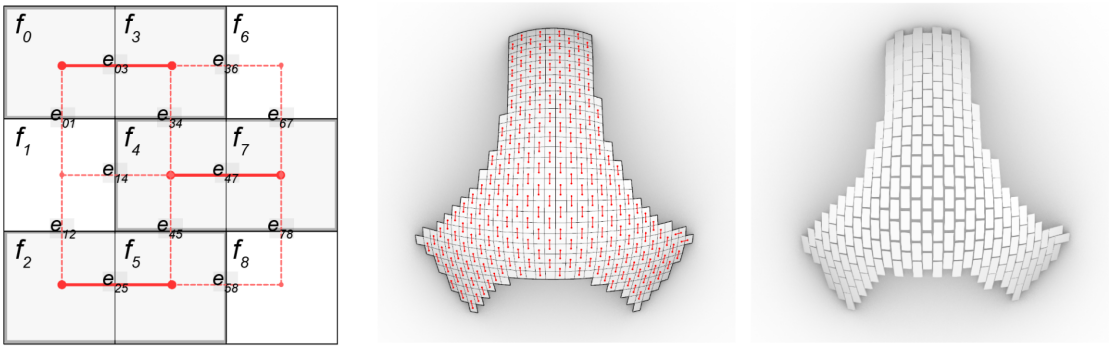


Figure 6. Running bond pattern and the connected face pairs; middle and right: running bond pattern outcome using quad mesh from Figure 3. right.

3.5 Generate tiles

After the matching process, the tile geometries can be generated using the facial information of the matched faces. We establish the origin of the tile by determining the midpoint of the line that connects the two centroid points of the face, and consider the direction of the line as the tile's length direction. Next, we compute the best-fit plane of all the vertices on the two faces, with the plane's normal being used as the thickness direction of the tile. By calculating the cross product of these two directions, we obtain the height direction, allowing us to construct the tile using these three directions. We implemented this step in Rhino 7 using the COMPAS framework [22]. The results are displayed in Figure 3. and Figure 6., right.

4. Results and discussion

4.1 Results

We apply the tile laying method explained in section 3 to the doubly curved free-form shells shown below. Our method produces satisfactory results for the shallow shell with small curvature (as depicted in Figure 7.), accurately computing the quadrilateral mesh and staggered pattern without any noticeable gaps or overlaps. However, for the shell shown in Figure 8., taking the geometry of [8], we observe larger gaps in regions with higher curvature, as explained in Section 3.1.

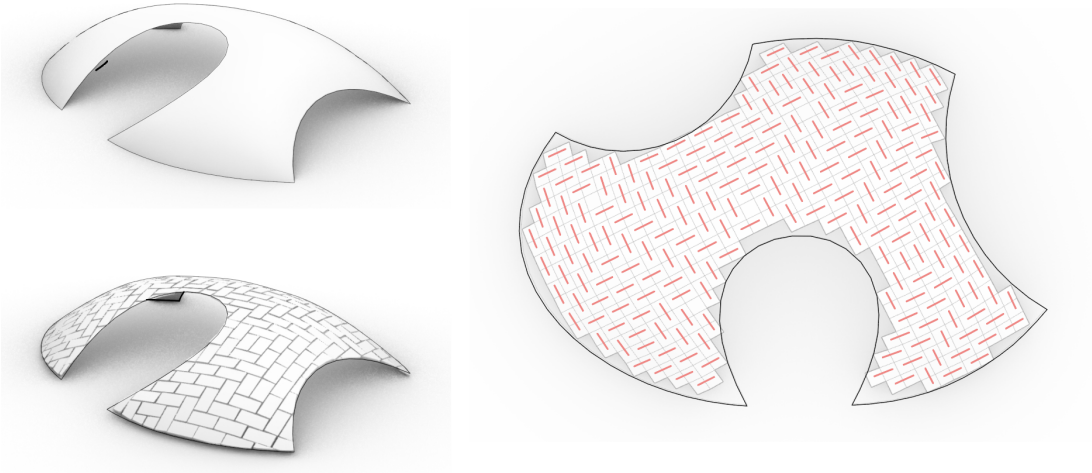


Figure 7. input mesh, tiled surface with irregular tiles at the boundary, and quadrangulation and matching outcome of a shallow shell.

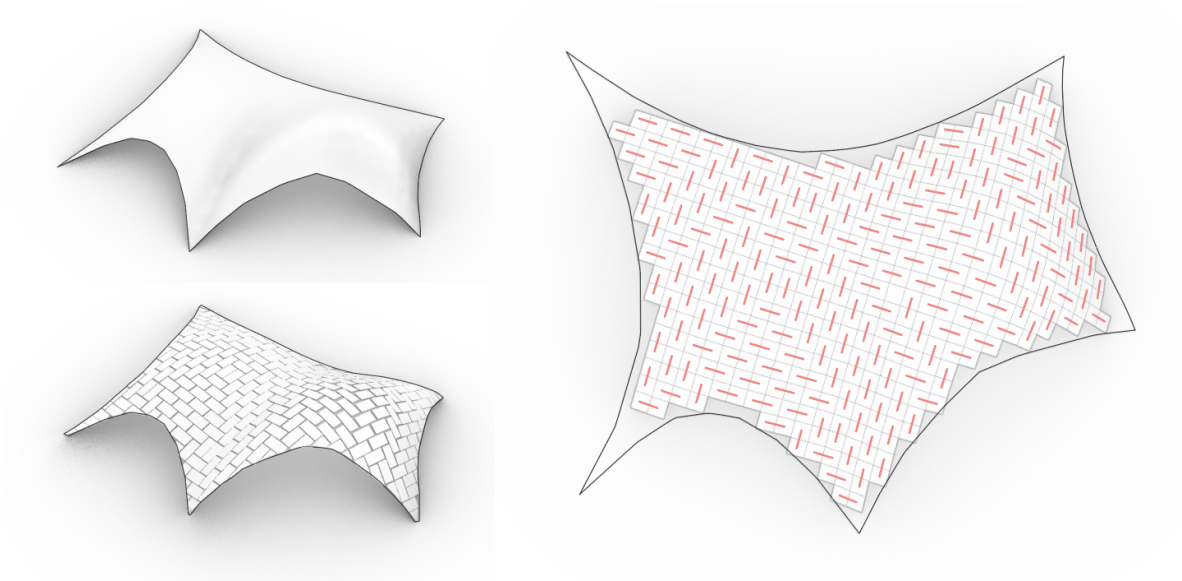
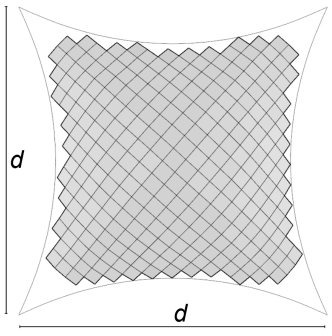


Figure 8. input mesh, tiled surface with irregular tiles at the boundary, and quadrangulation and matching outcome of a shell with higher curvature.

4.2 Performance

To evaluate the performance of our method in section 3, we conduct tile patterning on a shell with dimensions (d) as shown in Figure 9. left. The shell is supported at its four corners and tessellated with tiles measuring 280 × 140 × 15 mm. We ensure that all the quadrilateral faces have edge lengths greater than 140 mm. The dimension of the shell and the number of the faces are shown in Figure 9. right. The experiments are performed on a MacBook Pro (2020) with macOS 13.2.1, Apple M1 CPU and 8 physical cores. The computing time for section 3.1 is less than 1 second, and the computing time for section 3.2 using Gurobi Optimiser v10.0.0rc2 is listed in Figure 9. right. The time is the total (real) time reported by the time command. Our results indicate that the computing time grows exponentially with the input size. For a shell covering an area of approximately 50 m^2, the computing time is about 3 minutes.



Shell dimension	Number of faces	Time (section 3.2)
1.5 m	89	0.01 sec
2.5 m	185	0.20 sec
3.0 m	270	1.29 sec
4.5 m	531	8.85 sec
5.8 m	1192	30.58 sec
7.2 m	1679	183.19 sec

Figure 9. left: The input shell geometry; right: comparison of different shell dimensions, number of faces derived from section 3.1, and the corresponding computing time for section 3.2.

4.3 Contribution

Our approach tackles tile laying as a three-step global problem instead of a sequence of local problems. The presented approach allows us to adapt to different laying patterns by modifying the objective function in the matching step (as described in Section 3.4). Compared to other state-of-the-art methods, our approach can significantly reduce the need for customised cutting in the tiling pattern, particularly for smaller shell surfaces where the curvature is high in comparison to the tile dimension. For nearly developable surfaces, our approach can almost automate the tile-laying process, with little decision-making from the designers. For other free-form surfaces, our approach requires the designer to run an optimisation step after the variable quadrangulation to find a tileable mesh or separate the input surface into more developable patches.

4.4 Discussion

Our method relies purely on geometry and topology. We can compute different design solutions by changing the direction of the orientation field, such as changing it by 45 degrees or modifying the objective function in our matching step. However, our method cannot avoid distortion at the singularities, and in most cases, the resulting quadrangulation cannot align with the input surface boundary. The responsibility of balancing constructability and aesthetics still lies with the designer.

Additionally, solving the matching problem using a binary integer program solver can be prohibitively expensive for larger inputs (as discussed in Section 4.2). To deal with larger input sizes, one can split the input into smaller parts and solve the problem one part at a time.

Finally, our method does not consider structural behaviour. If we extend this method for tessellation patterns in masonry structures, there is potential for improvements by aligning the orientation field with the direction of force flow within the structure.

Acknowledgement

This research is supported by the A/T doctoral fellowship at ITA, ETH Zurich. Thanks to Ziqi Wang for the discussions and the suggestion for using Shape-Up for the shape optimisation step. Thanks to Stephan A. Kollmann for the discussion about the binary integer program, and correcting the notations in Section 3.

Code

Source code for reproducing the case studies is available from the corresponding author upon request.

Reference

[1] D. López, T. Mele, and P. Block, “Tile vaulting in the 21st century.,” *Inf. Constr.*, vol. 68, p. 162, Dec. 2016, doi: 10.3989/ic.15.169.m15.

[2] C. De Wolf, M. Ramage, and J. Ochsendorf, “Low Carbon Vaulted Masonry Structures,” *J. Int. Assoc. Shell Spat. Struct.*, vol. 57, no. 4, pp. 275–284, Dec. 2016, doi: 10.20898/j.iass.2016.190.854.

[3] D. López López, T. Van Mele, and P. Block, “The combination of tile vaults with reinforcement and concrete,” *Int. J. Archit. Herit.*, vol. 13, no. 6, pp. 782–798, Aug. 2019, doi: 10.1080/15583058.2018.1476606.

[4] M. W. A. Asali, “Craft-inclusive Construction,” p. 248.

[5] M. Ramage, “Guastavino’s Vault Construction Revisited,” *Constr. Hist.*, vol. 22, pp. 47–60, 2007.

[6] J. Ochsendorf and M. Freeman, “Guastavino Vaulting: The Art of Structural Tile,” Feb. 2010. Accessed: Mar. 31, 2023. [Online]. Available: https://www.semanticscholar.org/paper/Guastavino-Vaulting%3A-The-Art-of-Structural-Tile-Ochsendorf-Freeman./8c3f0fa2a96ab059a0441190450dc45c74a54850

[7] S. Rajabzadeh, “On the Computational Design of Free-form Masonry Vault,” 2015, doi: 10.6092/POLITO/PORTO/2616850.
[8] L. Davis, M. Rippmann, and T. Pawlofsky, “Innovative funicular tile vaulting: A prototype vault in Switzerland,” 2012.
[9] T. Bonwetsch and F. Gramazio, “Digitally Fabricating Non-Standardised Brick Walls,” 2007. Accessed: Mar. 31, 2023. [Online]. Available: https://www.semanticscholar.org/paper/Digitally-Fabricating-Non-Standardised-Brick-Walls-Bonwetsch-Gramazio/9e38ea121835b2eb5f8464cddf6f06bf215d4fd3#citing-papers
[10] S. Parascho, I. X. Han, S. Walker, A. Beghini, E. P. G. Bruun, and S. Adriaenssens, “Robotic vault: a cooperative robotic assembly method for brick vault construction,” *Constr. Robot.*, vol. 4, no. 3–4, pp. 117–126, Dec. 2020, doi: 10.1007/s41693-020-00041-w.
[11] D. Mitterberger *et al.*, “Augmented bricklaying: Human–machine interaction for in situ assembly of complex brickwork using object-aware augmented reality,” *Constr. Robot.*, vol. 4, no. 3–4, pp. 151–161, Dec. 2020, doi: 10.1007/s41693-020-00035-8.
[12] S. Rajabzadeh and M. Sassone, “Brick Patterning on Free-Form Surfaces,” *Nexus Netw. J.*, vol. 19, no. 1, pp. 5–25, Apr. 2017, doi: 10.1007/s00004-016-0305-9.
[13] E. Adiels, M. Ander, and C. Williams, “Brick patterns on shells using geodesic coordinates,” 2017.
[14] D. Panozzo, P. Block, and O. Sorkine-Hornung, “Designing unreinforced masonry models,” *ACM Trans. Graph.*, vol. 32, no. 4, pp. 1–12, Jul. 2013, doi: 10.1145/2461912.2461958.
[15] M. Eigensatz *et al.*, “Case Studies in Cost-Optimized Paneling of Architectural Freeform Surfaces,” in *Advances in Architectural Geometry 2010*, Vienna, 2010, pp. 49–72. doi: 10.1007/978-3-7091-0309-8_4.
[16] M. Eigensatz, M. Kilian, A. Schiftner, N. J. Mitra, H. Pottmann, and M. Pauly, “Paneling architectural freeform surfaces,” *ACM Trans. Graph.*, vol. 29, no. 4, p. 45:1-45:10, 2010, doi: 10.1145/1778765.1778782.
[17] H. Pottmann, M. Eigensatz, A. Vaxman, and J. Wallner, “Architectural geometry,” *Comput. Graph.*, vol. 47, pp. 145–164, Apr. 2015, doi: 10.1016/j.cag.2014.11.002.
[18] S. Bouaziz, M. Deuss, Y. Schwartzburg, T. Weise, and M. Pauly, “Shape-Up: Shaping Discrete Geometry with Projections,” *Comput. Graph. Forum*, vol. 31, no. 5, pp. 1657–1667, Aug. 2012, doi: 10.1111/j.1467-8659.2012.03171.x.
[19] D. Bommes *et al.*, “Quad-Mesh Generation and Processing: A Survey,” *Comput. Graph. Forum*, vol. 32, no. 6, pp. 51–76, 2013, doi: 10.1111/cgf.12014.
[20] J. Huang, Y. Zhou, M. Niessner, J. R. Shewchuk, and L. J. Guibas, “QuadriFlow: A Scalable and Robust Method for Quadrangulation,” *Comput. Graph. Forum*, vol. 37, no. 5, pp. 147–160, Aug. 2018, doi: 10.1111/cgf.13498.
[21] A. Jacobson *et al.*, “libigl: A simple C++ geometry processing library,” 2013, Accessed: Mar. 31, 2023. [Online]. Available: https://opus.lib.uts.edu.au/handle/10453/167463
[22] “COMPAS.” https://compas.dev/index.html (accessed Nov. 30, 2022).