Beautification of Islamic Patterns via Constraint Satisfaction

Yongquan Lu* and Erik D. Demaine
MIT Computer Science and Artificial Intelligence Laboratory
32 Vassar St., Cambridge, MA 02139, USA
yqlu@mit.edu, edemaine@mit.edu

Abstract

Islamic star patterns are traditionally drawn with compass-and-straightedge constructions, which yield geometrically precise results, but are often unintuitive to work out. We present a more declarative paradigm for construction, where users can draw patterns by hand and then apply constraints to improve and beautify the design. This system applies concepts from CAD systems like geometric constraint satisfaction and numerical optimization in a highly symmetric and aesthetic context. We exhibit some representative results generated by our system.

Background and Overview

Traditional Islamic ornament features intricately interlocking patterns with complex geometries and symmetries. Construction and design of these patterns was historically a closely guarded secret passed down from craftsman to apprentice, but some scholars have worked out compass-and-straightedge reconstructions for many classical Islamic designs [1]. However, while these procedures successfuly construct the original designs, they can be unintuitive and the reference points for the constructions may not always be obvious from the final design.

Instead of this imperative approach, we can take a *declarative* approach to Islamic geometric design. Extending on work like Kaplan's Taprats [2], we previously built Sliceform Studio [3] as a graphical interface where users inscribe patterns within tiles and drag tiles to create tilings. This allows artists to rapidly translate the pattern in their head to one on the screen, but drawing with a mouse is imprecise. Manually-drawn designs often contain more imperfections than those drawn with compass and straight-edge — for example, lines which were intended to be parallel might end up diverging slightly.

In this paper, inspired by similar features in CAD systems like Solidworks, we aim to build a system where users can declare various geometric constraints that their hand-drawn pattern should obey. The system resolves the constraints by transforming them into a numerical optimization problem. The vertices of the pattern yield degrees of freedom (i.e. decision variables), while the geometric constraints can be converted into an objective function, which we can seek to minimize over the state space with numerical optimization techniques.

Our strategy for optimization is as follows:

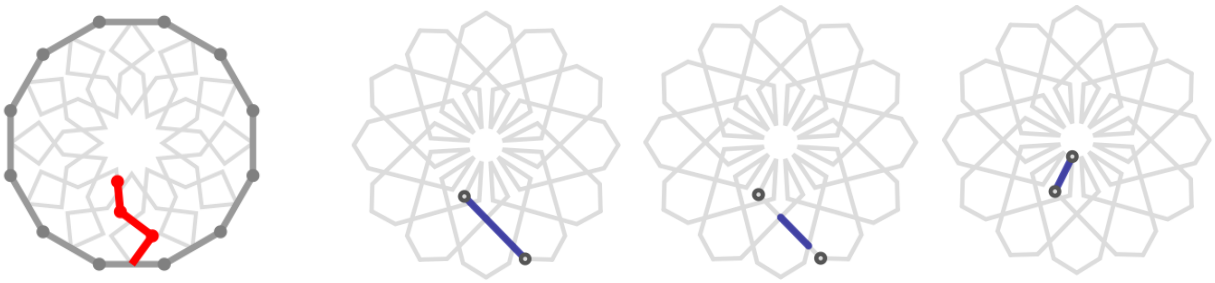
1. Specify segments in the design that should obey some constraint.
2. Identify the control points that generate the given segments.
3. Construct an objective function based on the constraint.
4. Vary the control points from (2) to minimize the objective function from (3).

The next few sections will discuss each of these steps in detail.

Segments and Control Points

In the vocabulary of this paper, *control points* are a set of points with coordinates specified by the user; Sliceform Studio extends them symmetrically to produce patterns in a tile. A *pattern* is a piecewise linear path that starts and ends on the boundary edges of a given polygonal tile. A *segment* is a linear subcomponent of a pattern that starts and ends with either a vertex or an intersection point.

When the user selects a segment, Sliceform Studio identifies the control points that generated that segment, and asks the user to explicitly identify the ones that are allowed to vary. This is because pattern optimization is often *directed* — for instance, the user might want to vary A but holding B, C and D constant so that $\overline{AB}$ is parallel to $\overline{CD}$.



(a) Three control points (in red) generate the full symmetric pattern.

(b) Examples of valid segments (shaded in blue) and their corresponding control points (circled). Note that the control points may or may not coincide with the endpoints of the segment.

Figure 1

Suppose there are n control points, that is, $2n$ real-valued degrees of freedom. We can construct a function `updateControlPoints` that takes in a vector in $\mathbb{R}^{2n}$, updates the corresponding coordinates of the control points and returns a tile configuration with regenerated patterns.

Constraints and Objective Functions

We have constructed a number of constraints that we have found to be sufficient to accomplish most optimization tasks in practice. Each of these constraints has a corresponding non-negative penalty function, where the constraint's penalty function equals zero if and only if the constraint holds.

Define $\angle(\overline{AB})$ as the acute angle $\overline{AB}$ makes with the horizontal. Here are a few of the most common constraints:

Constraint	Penalty Function
Parallel	$\left\lvert \angle(\overline{AB}) - \angle(\overline{CD}) \right\rvert$
Equal Length	$\left\lvert \lvert\lvert\overline{AB}\rvert\rvert - \lvert\lvert\overline{CD}\rvert\rvert \right\rvert$
Bisect	$\left\lvert \lvert\lvert\overline{AE}\rvert\rvert - \lvert\lvert\overline{EB}\rvert\rvert \right\rvert + \left\lvert \lvert\lvert\overline{CE}\rvert\rvert - \lvert\lvert\overline{ED}\rvert\rvert \right\rvert$ where $\overline{AB}$ and $\overline{CD}$ intersect at E

If multiple constraints are specified, we sum up each of the individual penalty functions to yield a global

objective function `evaluateObjective`. This might seem dimensionally problematic — how can we add up penalty functions if some of them return lengths while others return angles? However, if we merely see each penalty function as returning dimensionless quantities that happen to go to 0 as the constraint is fulfilled, there is no theoretical difficulty.

Most of the time, some solution exists where the multiple specified constraints can be satisfied simultaneously. However, if the constraints are incompatible (e.g. setting the same pair of segments to be both parallel and perpendicular), the minimum of the global objective function may not set all or any of the constituent penalty functions to 0.

To address this, Sliceform Studio allows the user to specify multiplicative coefficients (by default 1) to be applied to each constraint when summed up to yield the global objective. This gives the user finer-grained control to bias the optimizer towards solving one constraint in favor of a second incompatible one.

Numerical Optimization

We compose `updateControlPoints` and `evaluateObjective` to yield `optFunc`, a function that takes in a vector from $\mathbb{R}^{2n}$ and returns a real-valued objective to be minimized. The problem is reduced to a purely numerical one — we can then pass `optFunc` and the initial value of the state vector into one of many nonlinear optimization procedures (for example, conjugate gradient descent).

Quick experiments, however, motivated two further corrections to the `evaluateObjective` function:

1. Sometimes, varying a control point too far from its original position changes the topological relationships of the patterns — for example, causing two patterns that didn't previously intersect to now cross. To prevent this, we model each pattern vertex in the tile as an electrostatically charged particle that repels every other vertex. The sum of the repulsion force between every pair of vertices is added as a correction term in `evaluateObjective` and prevents any two vertices from coming too close to each other during optimization.
2. Sometimes, the numerical optimization returns a solution where the constraints are technically satisfied but control points lie outside the tile. This is obviously undesirable, and we resolve this by introducing a small penalty term for control points close to the boundary of the tile, and a larger penalty term for control points outside the tile.

(a) Pattern vertices may cross each other during optimization, causing the pattern structure to be qualitatively different.

(b) Optimization may move control points beyond tile boundaries.

Figure 2: Two types of analomous behavior that can arise during numerical optimization; we avoid them by introducing correction terms to the objective function.

Choice of optimization procedure

The two correction terms above are not differentiable; when added to `evaluateObjective`, nonlinear optimization methods like conjugate gradient descent that rely on a numerical gradient are no longer applicable. Hence, we have opted to use Powell's method [4], an algorithm that searches in the n-dimensional space for the minimum along a set of n direction vectors that are updated according to the past displacements. This algorithm is ideal as it does not require taking derivatives.

Results and Discussion

We find that pattern optimization works very well in practice and is able to beautify a wide range of hand-drawn designs. Indeed, the existing flexibility of Sliceform Studio's pattern design capabilities coupled with this new optimization functionality are able to rapidly replicate many classical designs, for which previously only lengthy compass-and-straightedge constructions were known. A simple example is shown below.

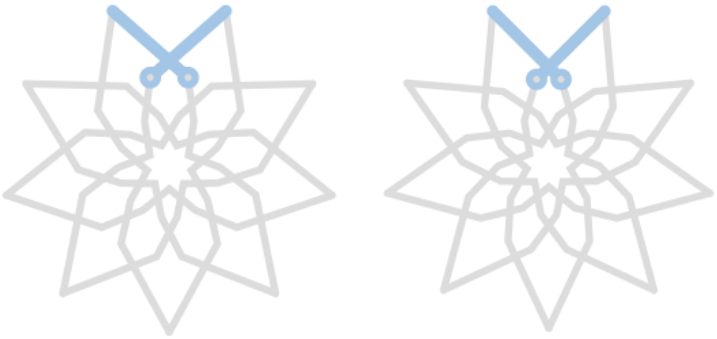


Figure 3: Before and after the highlighted pair of segments are constrained to be perpendicular. The circled controlled points indicate the degrees of freedom.

A more extensive video showcase of the entire workflow is available at `https://youtu.be/60PqB_Gh-nE`. The reader is invited to try out the system at `http://www.sliceformstudio.com/app.html?template=optimize_example`.

One area for future improvement is performance — the optimization process currently takes anywhere between 1 and 10 seconds. We also hope to eventually develop heuristics that allow the system to automatically find and correct small imperfections, instead of requiring users to explicitly specify constraints.

We thank Craig Kaplan for early discussions that influenced the development of this work.

References

[1] Eric Broug. *Islamic Geometric Design*. Thames & Hudson, 2013.

[2] Craig S Kaplan. Computer generated islamic star patterns. In *Proceedings of Bridges 2000*, pages 105–112, 2000.

[3] Yongquan Lu and Erik D. Demaine. A system for generating paper sliceform artwork. *Symmetry: Culture and Science*, 26(2):203–215, 2015.

[4] M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2):155–162, January 1964.