Animated Isohedral Tilings

Craig S. Kaplan

School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada
csk@uwaterloo.ca

Abstract

Inspired by abstract mathematical animated loops like those created by David Whyte (@beesandbombs), I explore animations based on isohedral tilings. I present a complete interactive tool for designing tiling animations, and show a few results that I have obtained using this tool.

Introduction

Animated GIFs are a popular means of representing short segments of silent, looping animation. This popularity is something of an anomaly—the format has long been superseded by other more capable types of digital video. The longevity of animated GIFs can be explained by the fact that they were one of the first (and for a time, probably the only) convenient means of representing and transmitting moving images on the internet. Arguably they are also favoured because of their relatively compact file size, their widespread support on social media sites, and their indelible association with memes. They are perhaps the closest modern-day equivalent to zoetropes, simple animation viewers that were popular in the 19th century.

Animated GIFs are also popular in a small community of digital artists who produce looping animations of geometric landscapes or abstract forms. Here I am most inspired by David Whyte, known as "beesandbombs" on social media (twitter.com/beesandbombs). His animations are usually of purely abstract geometry, and often contain clever tricks by which the the motion can be made to return to its starting point, creating a seamless loop ideal for the medium. Other artists working in this style include Paolo Ceric (patakk.tumblr.com) and David Szakaly (dvdp.tumblr.com). Andreas Wannerstedt (www.andreaswannerstedt.se) produces what he calls "oddly satisfying" loops, which are conceptually similar though his visual style is more figurative and photorealistic.

Inspired by this visual style, in 2018 I began to create a few of my own looping animated GIFs. These were based on tilings of the plane where the tile shapes evolved according to a few simple rules. Each animation was described by writing a short program from scratch in Processing. The results were satisfying, but the programming effort was too great to sustain.

In this paper I offer a more complete solution: an interactive tool for authoring animated tilings. The animations that I had previously programmed by hand can instead be designed easily via a user interface, greatly streamlining the process. I focus on the *isohedral tilings*, a family of simple shapes that can be manipulated easily in software. As a proof of concept, I also focus on a narrow world of possible animations, but one in which interesting (and perhaps, oddly satisfying) animations may nevertheless be found.

Animated Tilings

The idea of animating a tiling is not new. The artist M.C. Escher produced numerous "regular divisions of the plane" [9], which inspired generations of artists and mathematicians to explore figurative tilings and animations based on them. The earliest Escheresque computer-animated tiling may be "Symmetry Test 11A", created by Paul Allen Newell in 1982. The most compelling contemporary animations are those by Makoto

Nakamura (tessella.sakura.ne.jp/animation.html). For the most part they use an effect reminiscent of Escher's *Sky and Water*: shapes emerge from the tiling, perform a motion without being restricted by the constraint of fitting together in the plane, and then recede back into the tiling. Kevin Lee has explored various techniques for evolving the edges in a tiling [8], with the changes being rendered both in time (as an animation) and in space (as in Escher's *Development I*).

On the more abstract side, Nick Thompson's "Glambient" (nixweb.com/glambient) was a general-purpose tool for creating animations of arbitrary periodic tilings. It arguably operated at a low a level of abstraction, requiring a great deal of manual labour from the animator. More recently, Roice Nelson has been posting abstract diagrams of Euclidean and non-Euclidean tilings as @TilingBot on Twitter. In some cases, the tilings are smoothly translated or rotated in a seamless loop.

Instead of viewing an animated tiling as a sequence of images changing in time, we can evolve the shapes of tiles along a direction of space. Here we may draw upon two clear sources of inspiration. Escher was a master of this form; as I have explained elsewhere [3], he used a number of visual "metamorphosis" devices to draw tilings that change spatially. Inspired by Escher, the architect and designer William Huff developed "parquet deformations" [2], which were designed to be more abstract and geometric exercises. I have explored several techniques for drawing parquet deformations [3, 5], some of which are relevant here. Temporal animations of tilings are arguably easier to construct than spatial animations, because in the latter case the tiles are changing their shapes in the same dimension in which they are trying to interlock.

The Isohedral Tilings

In order to explain the workings of an animation system for isohedral tilings, I must first present enough mathematical background that the terms used in the rest of the paper may be understood. Tiling theory is a broad field within mathematics, and this section is necessarily superficial and incomplete. Grünbaum and Shephard offer the definitive treatment of the subject [1]; my own book [4] might also provide a useful introduction, along with extra details on algorithms and data structures for manipulating isohedral tilings in software.

Let S be a simple shape in the plane, a topological disc bounded by a Jordan curve. A *monohedral tiling* is a countable set of shapes $T_1, T_2, \ldots$, each a congruent copy of S, which cover the plane with no gaps (that is, the union of the T_i is the whole plane) and no overlaps (no point in the plane lies in the interior of more than one tile). We refer to S as the *prototile* of the tiling. The tiles naturally decompose the plane into a kind of planar subdivision, with vertices where three or more tiles meet joined by paths that run along the shared boundaries of two neighbouring tiles. We refer to these vertices as *tiling vertices*. If S is a polygon, its polygon vertices may or may not coincide with these tiling vertices; a *shape vertex* of a tile T_i is a polygon corner that is not also a tiling vertex.

As a drawing in the plane, a tiling will naturally have a group of symmetries, rigid motions that map the whole tiling to itself. Given tiles T_i and T_j, a congruence that maps T_i to T_j may or may not be a symmetry of the tiling; if it is, we say that T_i and T_j are *equivalent*. When all the tiles are equivalent in this manner, the tiling is called *isohedral*. Every isohedral tiling is necessarily *periodic*: its symmetry group is one of the 17 wallpaper groups.

Grünbaum and Shephard showed that the structure of an isohedral tiling can be distilled down to an *incidence symbol*, a compact symbolic description that summarizes the relationships between a tile and its neighbours. Every isohedral tiling has one of only 93 distinct symbols, which thereby partition the isohedral tilings into 93 tiling types **IH1**, **IH2**, . . . , **IH93** . Each type represents a family of prototiles that have similar "personalities": they lead to tilings with the same symmetries and topological structures, and can be manipulated in the same ways. Figure 1 shows an example of several tilings all belonging to type **IH21**.

From an artistic point of view, the isohedral tilings occupy a sweet spot in the trade-off between

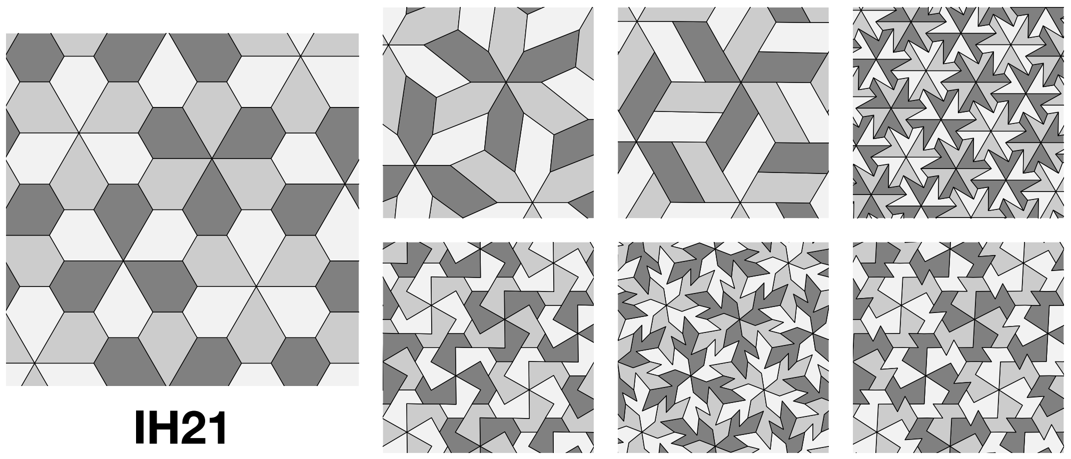

Figure 1: *Exploring a space of tilings. All tilings are of isohedral type* **IH21***, with a canonical representative shown on the left. On the right, the top row shows tilings that can result from modifying the tiling vertex parameterization and keeping the tiling edges straight, and the bottom row shows tilings with the same tiling vertex configuration as the example on the left but varying positions of shape vertices.*

complexity and expressiveness: they are flexible enough to describe a large number of interesting shapes—nearly all of Escher's monohedral tilings are also isohedral [9]—while still being simple enough to encode conveniently in software.

Previously I developed such an encoding as a basis for automatically constructing tilings in the style of Escher's regular divisions of the plane [6]. In this representation, the boundary of the prototile S is decomposed into the positions of the tiling vertices (which must be same for every copy of S in the final tiling) and the shapes of the paths that join them. The positions of the tiling vertices are determined by a small number of degrees of freedom that I called the *tiling vertex parameterization*. The paths are transformed copies of a small number of distinct path shapes; this inherent redundancy allows them to interlock when tiles are assembled.

I recently built a new version of my old library, based on lessons learned in nearly 20 years of using it. This library, called Tactile, offers an efficient, lightweight means of representing and manipulating isohedral prototiles and tilings in software. It is available for C++ (github.com/isohedral/tactile) and Javascript (github.com/isohedral/tactile-js). Part of the motivation for this paper was to develop new software that uses the Tactile library in interesting ways.

A Tiling Animation Tool

Using the Tactile library mentioned in the previous section, I developed an interactive authoring tool for creating animated isohedral tilings. The tool is based on *keyframe animation*, a well known approach used in both hand-drawn and computer animation. In traditional cel animation, a lead animator would plan out a shot by drawing a sparse set of "keyframes", the animation frames most crucial in communicating an action. A team of "inbetweeners" would then draw all the intermediate frames needed to produce smooth motion between the keyframes. In computer animation, a keyframe records the values needed to describe an object (for example, a character's joint angles). The computer can generate intermediate frames by interpolating between keyframe values. In practice, virtual characters in animated films are typically posed by hand in nearly every frame—interpolation is not considered sufficiently expressive. For my abstract animations, however, simple forms of interpolation are sufficient.

In keeping with the film analogy, an animation in my software consists of a contiguous sequence of *shots*. A shot is an interval of time with attached keyframes, which determine the configuration of the tiling for the duration of that interval. The keyframes within a shot must all have the same isohedral tiling type, thereby avoiding the thorny mathematical challenge of interpolating between fundamentally incompatible tilings. It is the animator's responsibility to manage continuity at the moment when one shot ends and the next begins, by ensuring that the tilings drawn immediately before and after that moment coincide.

A single keyframe offers a large number of degrees of freedom that describe a tiling's shape and structure, and that are subject to interpolation within a shot. Here I provide additional details on the features of a tiling that my software can animate, and any special considerations required for these features.

As discussed above, every isohedral tiling type comes equipped with a tiling vertex parameterization: a small set of real-valued parameters that control the positions of the prototile's tiling vertices. The Tactile library maintains a set of parameters in its description of a tiling. Every keyframe records specific values for these parameters, and intermediate values can be computed via simple linear interpolation. In particular, if a given parameter has value p_0 at time t_0 and value p_1 at time t_1, then for any time $t_0 < t < t_1$ we can set $p = (1 - f)p_0 + fp_1$, where $f = (t - t_0)/(t_1 - t_0)$.

Tactile makes no assumptions about the paths that will be drawn to connect the tiling vertices, allowing users of the library to use whatever geometric primitives they like. Because my goal here is to create abstract geometric animations, I use simple piecewise-linear paths (i.e., line segments joined end-to-end) for the tile's boundary. Furthermore, I require that for every keyframe in a shot, tile edges all have the same number of free vertices. This restriction greatly simplifies the problem of interpolating between paths associated with different keyframes: I linearly interpolate the x and y coordinates of the corresponding path vertices. Future work should explore more general paths, including perhaps circular arcs and Bézier splines. However, interpolating in an aesthetically pleasing way between two general paths is a difficult research problem; while several practical algorithms have been proposed in the computer graphics literature, no definitive solution exists.

I also augment the representation of a tiling with a global transformation, a combination of translation, rotation, and uniform scaling. The transformation is stored using four numbers: a translation vector (t_x, t_y), an orientation angle θ, and a scaling factor α. It would be inconvenient to store transformations as matrices, because linear interpolation produces incorrect results when applied directly to matrix entries. By default, rotation and scaling are applied relative to the origin in world coordinates. However, this point is arbitrary relative to Tactile's definition of the tiles in a tiling. I therefore define an "anchor point" (a_x, a_y) that serves as the origin for scaling and rotation operations. The anchor can be placed anywhere (and animated as well); when it is moved interactively, the interface will attempt to snap it to points of interest in the tiling (vertices, edge midpoints, and tile centres).

When animating, we can linearly interpolate the rotation angle as well as the coordinates of the anchor point and translation vector. However, the scaling factor requires more care. Let a shot have scale factors α_0 at time t_0 and α_1 at time t_1. The scaling function should change smoothly and monotonically from t_0 to t_1, but equal increments in time should correspond to equal *ratios* in scale, not *differences*. Thus I use logarithmic interpolation instead of linear, setting $\alpha_t = \alpha_0^{1-f}\alpha_1^{f}$, where $f = (t - t_0)/(t_1 - t_0)$, as before. Finally, if at a given time we have a translation (t_x, t_y), an orientation θ, a scaling factor α, and an anchor position (a_x, a_y), we construct the 3×3 homogeneous affine transformation matrix $T(t_x, t_y)T(a_x, a_y)S(\alpha)R(\theta)T(-a_x, -a_y)$, where T, S, and R correspond to standard matrix representations of translation, scaling, and rotation. Every point in the tiling is transformed by this matrix when it is drawn.

Sometimes it can be difficult to judge the correct transformation needed to bring two tilings into perfect coincidence at the boundary between two consecutive shots. My user interface provides tools to assist in constructing the correct transformation. I support a limited form of "onion skinning", a method from traditional animation in which reference frames are made visible underneath the current one. I allow the animator to choose any keyframe as a reference, with the current keyframe is drawn semi-transparently over

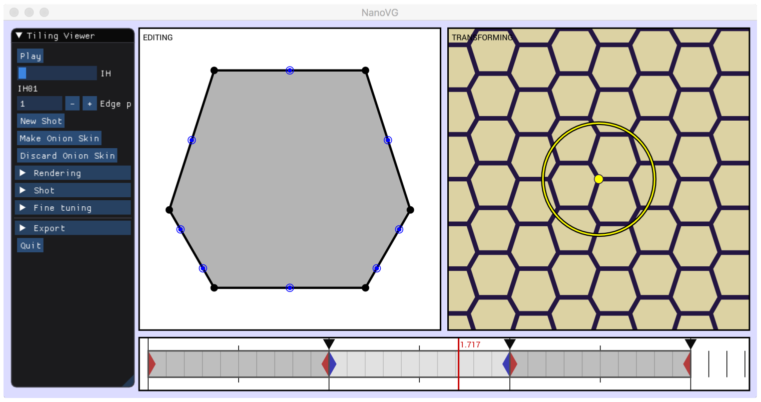

Figure 2: *A screenshot showing the animation system in operation. From left to right along the top, the window contains a console with controls and options, an editor for the current prototile, and a viewer/editor for the transformed tiling. The bottom contains a timeline divided into shots, with keyframes in red and blue.*

it. Often this superposition is enough to produce the correct transformation through interactive manipulation. When more precision is needed, it is possible to define a "reference segment" in each of the lower and upper layers. The reference segment can be drawn interactively, and its endpoints snap to features in the same way as anchor points. When needed, the upper tiling will be transformed so that its reference segment is brought into perfect alignment with that of the reference tiling.

In some cases, pure linear interpolation is too rigid to produce a pleasing animation. After all, real-world objects have inertia and cannot change abruptly from a resting state to motion at a constant speed. In many of Whyte's animation loops, individual motions accelerate to full speed and then decelerate to a stop. Accordingly, in my software every shot is assigned one of several *easing* modes to simulate acceleration at the beginning of the shot's motion, deceleration at the end, or both. Easing is implemented on top of pure linear interpolation by altering the times at which frames are sampled. Let $\omega(x)$ be an easing function, typically a monotonic function with $\omega(0) = 0$ and $\omega(1) = 1$, applied to a shot with start time t_s and end time t_e. Given a set of equally spaced time samples $t_s = t_0 < t_1 < t_2 \ldots < t_n = t_e$, we would apply easing by rendering frames at times $t_i' = (1 - f_i)t_s + f_i t_e$, where $f_i = \omega((t_i - t_s)/(t_e - t_s))$. Easing curves have become ubiquitous in motion graphics, web design, and user interface design, though they trace their lineage back to the early days of computer animation [7] in the form of the "slow-in, slow-out" principle, which was itself adapted from the core animation principles developed early on by Disney animators [10].

Developing keyframe animation software can require a significant coding effort, in particular because of the complexity of the user interface needed to support interactive authoring. My tiling animator comprises about 4000 lines of C++ code, and runs on top of the OpenGL graphics library with the help of a number of open-source utility libraries. Beyond my own Tactile library, I use GLFW (www.glfw.org) to create an interactive OpenGL window; nanovg (github.com/memononen/nanovg) for drawing vector graphics in OpenGL and Cairo (cairographics.org) for drawing to images and PDFs; Dear ImGui (github.com/ocornut/imgui) for widgets like buttons and sliders; Magick++ (imagemagick.org/Magick++) to export animated GIFs; and Niels Lohmann's JSON library (github.com/nlohmann/json) to define a file format for reading and writing animation files. Figure 2 shows a screenshot of the animator in operation.

Results

Armed with the system described in the previous section, it now becomes possible to explore the space of animated isohedral tilings in search of interesting or unusual cases. Ideally, this exploration can yield a few new design principles to guide the creation of effective animations.

Certainly, any value controlling the shape of a tile or the arrangement of tiles in the tiling can be varied periodically. If all such values vary with the same period, that cycle can be expressed in an animation consisting of a single shot whose start and end points are identical. However, these animations can seem obvious: the mystery of their construction is too easily resolved. It is much more interesting to search for loops that have the same uncanny aesthetic as the best of Whyte's animations. Here we need to find unexpected opportunities to close an animation onto itself, to leave behind a riddle that begs to be understood. I have found a few such devices in my explorations, and below I present examples of them.

In some cases, a prototile can be deformed to the point where its boundary meets itself in one or more locations, and pinches the tile's interior off into multiple regions that are congruent to the original shape. Consider for example the square prototile represented in the space of **IH73**, as shown in the top row of Figure 3. When opposing edge midpoints are brought into coincidence, the prototile splits into two smaller squares, each with side length $1/\sqrt{2}$ of the original, and rotated by 45°. If the tiling as a whole is rotated by 45° and scaled by $\sqrt{2}$ while the vertices are moving, it will arrive at a configuration congruent to the original just as the tiles are degenerating into pairs of squares. The resulting animation loop is fascinating: it zooms in forever without progressing towards an endpoint, in a manner reminiscent of the impossible staircase in Escher's *Ascending and Descending*. Other tiling types can yield similar loops; Figure 3 also shows two examples where the equilateral triangles of **IH90** are split first into threes, and then into fours.

A related situation can sometimes occur in which two adjacent edges of a tile collapse into a line segment. Figure 4 shows an example based on **IH71**. At the point of degeneracy, the tiling collapses into what appears to be 2×1 rectangles. If we render only the interiors of the tiles and not the edges, adjacent rectangles of the same colour will be perceived as joining together into squares, which once again can be used to loop back to the start of the animation under a suitable rotation and scaling.

Turning back to the topmost example of Figure 3, if we attempt a similar deformation with a hexagonal tiling type like **IH18**, a new opportunity emerges. At the point where shape vertices meet, the prototile splits not into copies of the original hexagon, but into three 60° rhombs. The rhombs combine to give a tiling of a new topology, which can be expressed using a different isohedral type such as **IH36**. These "topological transitions" can add excitement to an animation, as they provide a surprising change that might seem unlikely at first. We can then return to hexagons by reversing the change, or via a different route, as shown in Figure 5.

Many other topological transitions are possible using this system. A simple trick is to find settings for the tiling vertex parameterization that bring tiling vertices into coincidence. At such points, hexagons can be made to collapse into squares or triangles, for example. Figure 6 shows a "grand tour" that traverses the three regular tilings of the plane, smoothly transitioning between each pair. In previous work [3] I documented a number of topological transitions of this type, for use in the spatial case of parquet deformations; such transitions are equally applicable here.

Many other interesting animations are achievable with this system; please see the supplemental files for additional examples.

Conclusion

I have created a prototype animation tool that makes it possible to explore the intriguing world of animated isohedral tilings. The tool streamlines the process of creating new animations that otherwise might each have required a significant amount of custom programming.

There any many opportunities for improving the software. Most of the time, tiling vertices and shape vertices in keyframes must be positioned very carefully: to bring two points into coincidence, for example, or to divide a tile into recognizable shapes like squares. The interface should support more sophisticated "snapping" tools that can guess where the user is attempting to position points, so that the point can be warped to its mathematically correct position. For tiling vertices, this problem can be mathematically non-trivial,

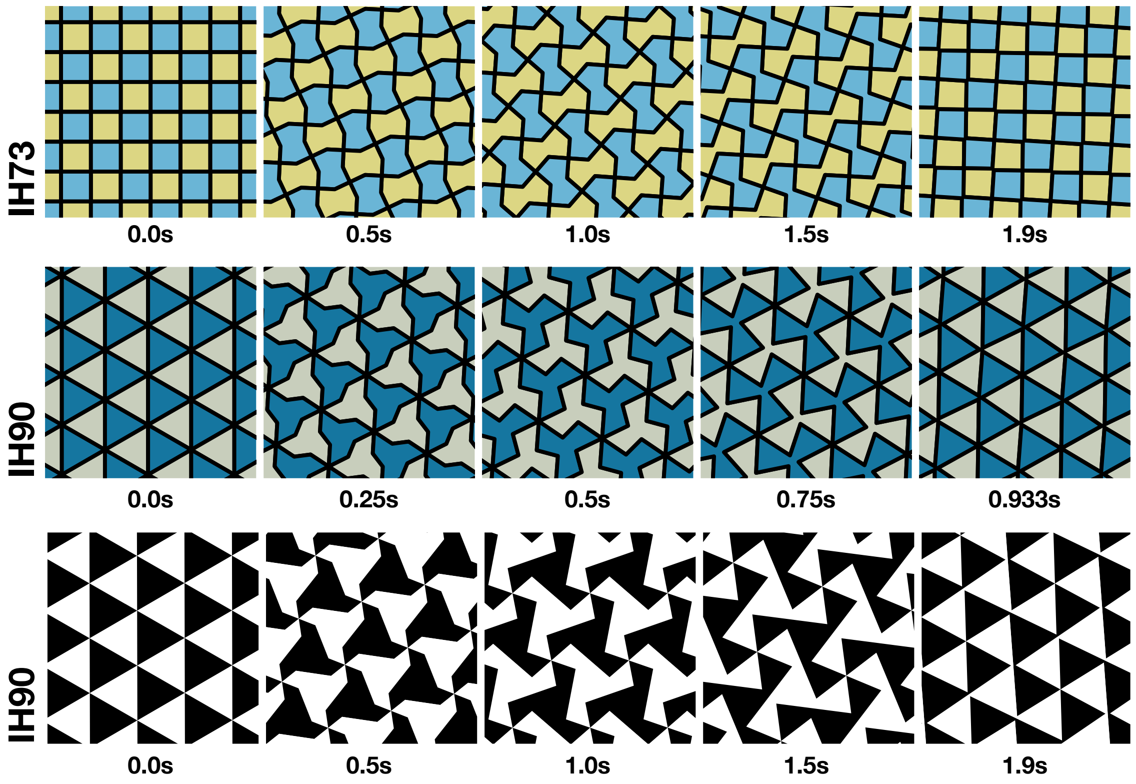

Figure 3: *Three animations in which a single tile is split into multiple smaller copies of itself, while the tiling transforms to bring the split copies into congruence with the original. In this paper, all animations are shown as snapshots; animations are provided in the supplementary files.*

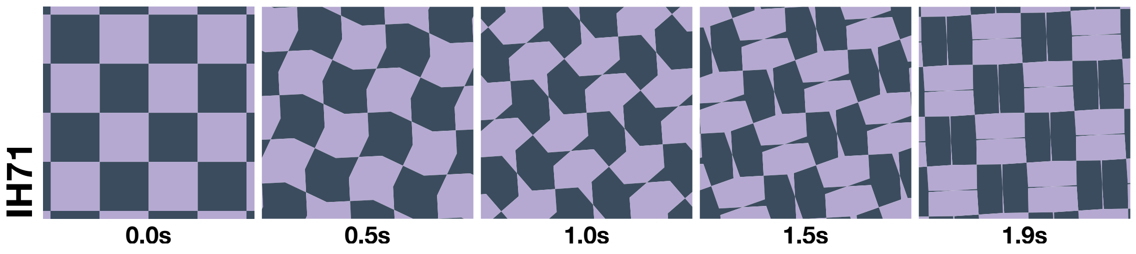

Figure 4: *An animation in which pairs of tiles merge along shared edges to form larger copies of the original square prototile, while the tiling zooms out and rotates to compensate.*

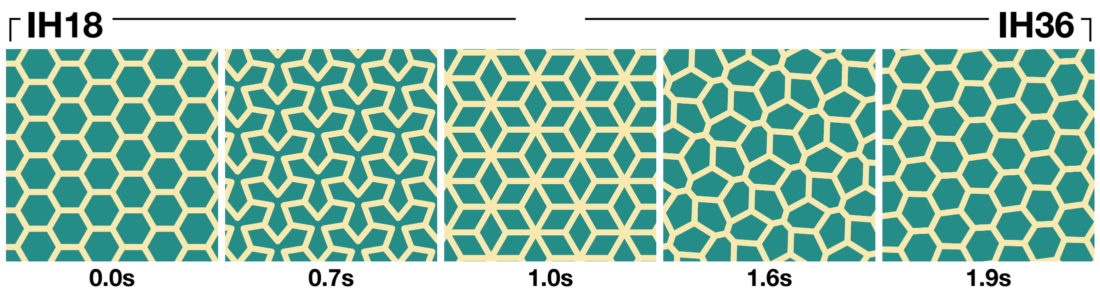

Figure 5: *An animation that includes a change of topological type. At the midpoint of the animation, hexagons degenerate into* 60° *rhombs, which can then be deformed back into hexagons.*

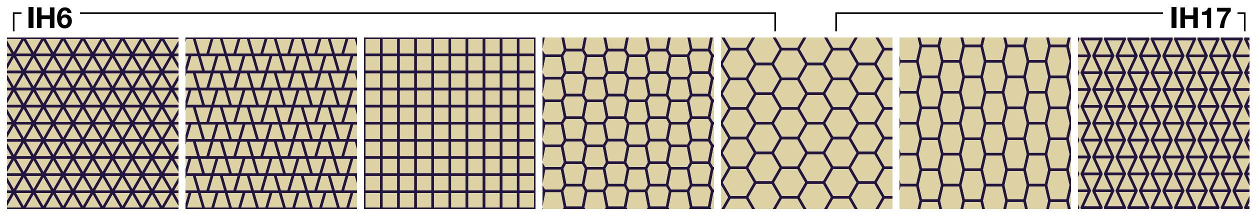

Figure 6: *A "grand tour" through the three regular tilings of the plane, with the help of two different isohedral tiling types.*

because they are coupled indirectly to the underlying tiling vertex parameterization.

The greatest challenge with this system is simply that it works best as a way to construct an animation that I can already visualize in my mind's eye. That is, the animator must already be quite familiar with the isohedral tilings in order to create compelling loops. It would be interesting to explore new interface paradigms that allow even novice users to explore this space of animations productively, or perhaps even computer algorithms for automating the discovery of these sorts of uncanny loops.

References

[1] Branko Grünbaum and G.C. Shephard. *Tilings and Patterns*. Dover, second edition, 2016.

[2] Douglas Hofstadter. *Metamagical Themas: Questing for the Essence of Mind and Pattern*. Bantam Books, 1986.

[3] Craig S. Kaplan. Metamorphosis in escher's art. In Reza Sarhangi and Carlo H. Séquin, editors, *Bridges Leeuwarden: Mathematics, Music, Art, Architecture, Culture*, pages 39–46, London, 2008. Tarquin Publications.

[4] Craig S. Kaplan. *Introductory Tiling Theory for Computer Graphics*. Morgan & Claypool, 2009.

[5] Craig S. Kaplan. Curve evolution schemes for parquet deformations. In George W. Hart and Reza Sarhangi, editors, *Proceedings of Bridges 2010: Mathematics, Music, Art, Architecture, Culture*, pages 95–102, Phoenix, Arizona, 2010. Tessellations Publishing.

[6] Craig S. Kaplan and David H. Salesin. Escherization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques (SIGGRAPH 2000)*, pages 499–510. ACM Press/Addison-Wesley Publishing Co., 2000.

[7] John Lasseter. Principles of traditional animation applied to 3d computer animation. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pages 35–44, New York, NY, USA, 1987. ACM.

[8] Kevin Lee. Algorithms for morphing escher-like tessellations. In Douglas McKenna Kelly Delp, Craig S. Kaplan and Reza Sarhangi, editors, *Proceedings of Bridges 2015: Mathematics, Music, Art, Architecture, Culture*, pages 483–486, Phoenix, Arizona, 2015. Tessellations Publishing.

[9] Doris Schattschneider. *M.C. Escher: Visions of Symmetry*. Harry N. Abrams, second edition, 2004.

[10] Frank Thomas and Ollie Johnston. *The Illusion of Life: Disney Animation*. Disney Editions, 1995.