Vector Graphics Animation with Time-Varying Topology

Boris Dalstein, Rémi Ronfard, Michiel Van de Panne

Vector Graphics Animation with Time-Varying Topology

Boris Dalstein*
University of British Columbia

Rémi Ronfard
Inria, Univ. Grenoble Alpes, LJK, France

Michiel van de Panne
University of British Columbia

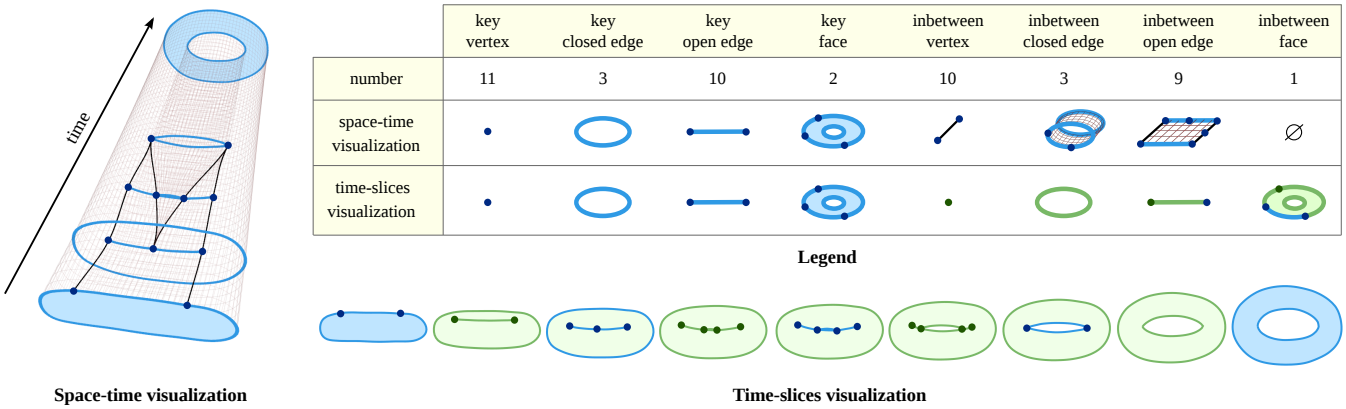


Figure 1: *A space-time continuous 2D animation depicting a rotating torus, created without 3D tools. First, the animator draws key cells (in blue) using 2D vector graphics tools. Then, he specifies how to interpolate them using inbetween cells (in green). Our contribution is a novel data structure, called Vector Animation Complex (VAC), which enables such interaction paradigm.*

Abstract

We introduce the Vector Animation Complex (VAC), a novel data structure for vector graphics animation, designed to support the modeling of time-continuous topological events. This allows features of a connected drawing to merge, split, appear, or disappear at desired times via keyframes that introduce the desired topological change. Because the resulting space-time complex directly captures the time-varying topological structure, features are readily edited in both space and time in a way that reflects the intent of the drawing. A formal description of the data structure is provided, along with topological and geometric invariants. We illustrate our modeling paradigm with experimental results on various examples.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: vector graphics, 2D, animation, topology, space-time, cell complex, boundary-based representation

*dalboris@cs.ubc.ca

1 Introduction

A fundamental difference between raster graphics and vector graphics is that the former is a *discrete* representation, while the latter is a *continuous* representation. Instead of storing individual pixels that our eyes readily interpret as curves, vector graphics stores curves that can be rendered at any resolution. As display devices spanning a wide range of resolutions proliferate, such resolution-independent representations are increasing in importance.

Similarly, a fundamental difference between traditional hand-drawn animation and 3D animation is that the former is *discrete* in time, while the latter is *continuous* in time. Instead of storing individual frames that our eyes interpret as motion, the use of animation curves allows a scene to be rendered at any frame rate.

Space-time continuous representations, i.e., representations that are resolution-independent both in the spatial domain *and* the temporal domain, are ubiquitous within computer graphics for their many advantages. They are typically based on the "model-then-animate" paradigm: a parameterized model is first developed and then animated over time using animation curves that interpolate *key values* of the parameters at *key times*. A limitation of this paradigm is the underlying assumption that the model can be parameterized by a *fixed* set of parameters that captures the desired intent. This is indeed possible for 3D animation and simple 2D animation, but it quickly becomes impractical in any 2D animation scenario where the number of strokes or how they intersect change over time. In other words, the "model-then-animate" paradigm fails when the *topology* of the model is time-dependent, which makes it challenging to represent space-time continuous animated vector graphics illustrations with time-varying topology.

In this paper, we address this problem by introducing the Vector Animation Complex (VAC). It is a cell complex immersed in space-time, specifically tailored to meet the requirements of vector graphics animation with non-fixed topology. Any time-slice of the complex is a valid Vector Graphics Complex (VGC) which make its rendering consistent with non-animated VGCs.

2 Related Work

Vector graphics Resolution-independent representations for 2D illustrations include stacked layers of paths (the most common), planar maps [Baudelaire and Gangnet 1989; Asente et al. 2007], diffusion curves [Orzan et al. 2008], stroke graphs [Whited et al. 2010; Noris et al. 2013], and vector graphics complexes [Dalstein et al. 2014]. We use the latter as a starting point for this paper, since unlike paths and diffusion curves, it is topology-aware (i.e., can represent shared boundaries between objects); unlike planar maps, it allows objects to overlap (which can be hard or impossible to avoid when interpolating key edges); and unlike stroke graphs, it can represent 2D faces (for coloring). Similarly to [McCann and Pollard 2009], we can achieve temporally local stacking orders.

Topology-unaware inbetweening Cartoon animation [Thomas and Johnston 1987; Blair 1994] consists in drawing a finite sequence of pictures that gives the illusion of motion. It was expected that automatic inbetweening of vectorized strokes would make cartoon animation easier, but this task appeared to be much more complex than expected [Catmull 1978], one reason being inconsistent *topology* between keyframes. Early stroke-based approaches [Burtnyk and Wein 1971; Reeves 1981; Fekete et al. 1995] are *manual* (the animator selects pairs of strokes to interpolate) and *topology-unaware* (strokes are interpolated independently, unaware of their neighbors). Recent methods [Liu et al. 2011; Yu et al. 2012] use shape descriptors and machine learning techniques to compute stroke correspondences automatically, but since they are topology-unaware, they are unable to generate space-time continuous animation with time-varying topology.

Topology-aware inbetweening [Kort 2002] introduces semantic relations between strokes (e.g., *intersecting*, or *dangling*), together with inference rules to find stroke correspondences automatically. Later, [Whited et al. 2010] introduces stroke graphs, where nodes are where strokes end or intersect, and edges are the strokes themselves. Given two stroke graphs and initial stroke correspondences, the two graphs can be traversed in parallel to propagate stroke correspondences, stopping at topological inconsistencies. Unfortunately, unlike our method, none of these methods can produce space-time continuous animations with time-varying topology, since it is not allowed by their representation. Also, none of these methods address coloring.

Data-driven inbetweening An alternative approach to generate cartoon animations is to reuse existing content. [Bregler et al. 2002] extracts animated affine transformations and weight coefficients from existing cartoons, which can be transfered to new shapes. [de Juan and Bodenheimer 2006] performs a semi-automatic segmentation of the input video to combine parts of existing content together. New inbetweens can be generated by defining an implicit space-time surface interpolating extracted contours, however, no change in topology is allowed, since interpolated contours are always closed curves. [Zhang et al. 2009] proposes a method to vectorize input cartoon animations, allowing to edit them. However, their outputs are stacked layer of paths, thus cannot represent topological events.

Shape morphing Another way to generate inbetweens is shape morphing, where a *shape* is a closed curve (its boundary), together with a raster image (its interior). To avoid shrinkage caused by naive solutions, [Sederberg et al. 1993] interpolates intrinsinc definitions of the shape boundary. [Alexa et al. 2000] interpolates compatible triangulations of the shapes by minimizing an as-rigid-as-possible energy, and uses texture blending for the interior pixels. It has been improved [Fu et al. 2005; Baxter et al. 2009], and extended to interactive shape manipulation [Igarashi et al. 2005]. Initial arc-length correspondences between the two closed curves can be achieved using curvature-based methods [Sebastian et al. 2003]. An alternative approach to shape morphing is introduced by [Sýkora et al. 2009], where they align the two shapes using an iterative method, which can be applied for temporal noise control [Noris et al. 2011], or texture transfer [Sýkora et al. 2011]. Unfortunately, none of these methods can produce space-time continuous animations of vectorized curves with changing topology, since by definition every shape has the topology of a disc, and their interior is not vectorized, typically leading to blurring artifacts.

Stylizing 3D animation A natural approach to handle image-space topological events is to animate in a different space where no topological events occur, e.g., 3D animation [Lasseter 1987], in which case a fixed number of degrees of freedom can be keyframed independently. From a 3D model, one can compute vectorized 2D feature lines (e.g., [Bénard et al. 2014]), from which it is possible to extract cycles for coloring using depth-ordered paths [Eisemann et al. 2009] or planar maps [Karsch and Hart 2011], which can be further processed in 2D for stylization. However, the 3D-to-2D conversion is typically a per-frame process and therefore does not output a time-continuous 2D animation. To address this issue, [Karsch and Hart 2011] tracks the snaxels' 3D positions in the original mesh to generate correspondences across 2D frames, [Buchholz et al. 2011] computes a parameterization of the space-time surface swept by the silhouette lines, and [Bénard et al. 2012] uses an image-space relaxation method to deform, split and merge active strokes at frame i to match the feature lines of frame $i + 1$. Unfortunately, unlike ours, all these methods require to create a 3D animation beforehand. In addition, their output representation either does not support vectorized coloring [Buchholz et al. 2011; Bénard et al. 2012], or breaks the animation into contour sequences that do not change in topology [Karsch and Hart 2011]. In this paper, we present a novel representation that could be used as output of these existing methods to address their limitations.

Using hybrid "2.5D" models To have better image-space control of style, but still animate in a space free from topological events, [Fiore et al. 2001; Rivers et al. 2010] introduce hybrid models where shapes are defined in 2D, but their interpolation and depth-ordering is guided by 3D information. Unfortunately, these approaches tend to reduce the space of possible animations (compared to freeform hand-drawn animation), and only allows the representation of topological events which are solved by depth ordering.

Space-time modeling Finally, another approach to generate 2D animations –the one adopted in this paper– is to consider animated lines as surfaces in space-time [Fausett et al. 2000; Kwarta and Rossignac 2002; de Juan and Bodenheimer 2006; Southern 2008; Buchholz et al. 2011], and animated faces as volumes in space-time [Fausett et al. 2000; Southern 2008]. Therefore, *animating* becomes *modeling in space-time*, which makes possible to easily represent topological events, unlike when using the model-then-animate paradigm. The time dimension can also be replaced by more abstract parameters [Ngo et al. 2000; Fausett et al. 2000], leading to 4D or even higher dimensional objects. Recently, space-time meshes have also be used for fluid simulation inbetweening [Raveendran et al. 2014]. In theory, any non-manifold topological representation could be used to apply these concepts, as long as they can represent objects of sufficiently high dimension. Simplicial complexes [De Floriani et al. 2010] are a natural choice for their simplicity and scalability in dimension. Other non-manifold representations that scale in dimension are G-maps [Lienhardt 1994]

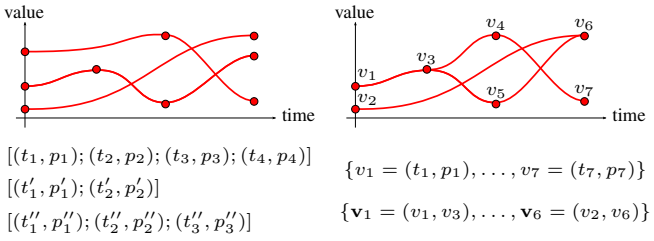


Figure 2: *Left: The existing keyframing paradigm, defining an animation as ordered sequences of key values. Right: Our more general approach, where key values are unordered but labeled, and inbetween values specify which one to interpolate.*

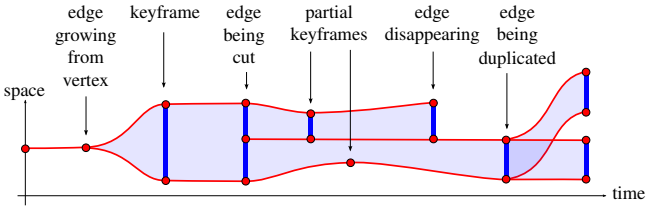


Figure 3: *Stroke graph animation with time-varying topology. Red dots are key vertices; (non-vertical) red curves are inbetween vertices; (vertical) blue curves are key edges; and light blue areas are inbetween edges. Each annotation describes either a topological event introduced by key cells, or specifies that key cells are used as conventional "keyframes" (trajectory control, no change in topology). Note: key edges are represented as straight lines (because space is represented as 1D), but are in fact general 2D curves.*

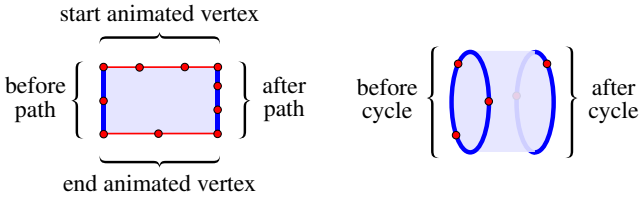


Figure 4: *Topology of inbetween edges. Note: a path or cycle can also consist of a single key vertex (but an animated vertex cannot). A cycle can also consist of a single closed edge, possibly repeated.*

and the selective geometric complex [Rossignac and O'Connor 1989]. If no more than three dimensions are needed, the radial-edge [Weiler 1985] or handle-cell [Pesco et al. 2004] structures could be appropriate choices as well. Unfortunately, none of these representations are *designed* to represent space-time objects, and while they can be very appropriate for algorithm processing, artistic space-time modeling using them is particularly non intuitive. What makes our representation unique is that it treats the time dimension separately from the space dimensions, enabling a keyframing paradigm similar to what animators are familiar with. For instance, instead of a 1D entity called "edge", we make the distinction between two types of 1D entities: a *key edge* (1D in space; 0D in time) represents an edge at a given time, and an *inbetween vertex* (0D in space; 1D in time) represents an interpolation between two key vertices. Even though they are both 1D in space-time, they are created, edited, and visualized differently, reflecting a cleaner semantics.

3 Space-Time Topology

In this section, we provide an initial intuition behind the vector animation complex, which we formally define in Section 4.

3.1 Animating vertices

Suppose an animator wants to create a time-continuous animation of a single vertex v. This means that he needs to define its position $p(t)$ for every time t in the life-span of the vertex. The existing approach (Fig. 2, Left) is to define a sequence of *keys* $[(t_1,p_1);(t_2,p_2);\ldots]$ which are interpolated in time. To animate three vertices, the animator would define three sequences of keys. Let us call this paradigm ***sequential keyframing***, since the representation is a set of sequences of keys: one sequence per animated vertex, or more generally, one per animated degree of freedom.

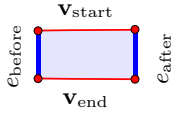
But what if the animator wants the number of vertices –or degrees of freedom– to change over time, by splitting or merging? We can observe (Fig. 2, Right) that the *space-time topology* of such animation is not anymore disconnected sequences, but a more general graph. Therefore, sequential keyframing fails to represent such animation with time-varying topology, and we need a more general approach to keyframing that we call ***topological keyframing***. The animator first defines a set of ***key vertices*** $v_i=(t_i,p_i)$, as in sequential keyframing except that they are not ordered in sequences. Then, he defines a set of ***inbetween vertices*** $\mathbf{v}_j=(v_{\text{before}},v_{\text{after}})$ that reference to two key vertices to interpolate.

In theory, such paradigm can easily be applied to animate any kind of values, say, quaternions. However, in this paper, we use it to animate the topology of vector graphics illustrations. This poses additional challenges due to the fact that such topology is already a graph-like structure in the space dimension. Therefore, we have to represent incidence relationships both in the temporal domain *and* the spatial domain, resulting in a space-time complex.

3.2 Animating stroke graphs

Suppose now that we want to animate a stroke graph [Whited et al. 2010], i.e. not only vertices but also (open) edges, which are 2D curves starting at a start vertex and ending at an end vertex. An easy way to achieve this is to define first a stroke graph, then use sequential keyframing to animate independently its degrees of freedom (e.g., position of the vertices and Bézier control points of the edges). Unfortunately, with this approach, it is impossible to represent animated stroke graphs with time-varying topology.

Our solution (Fig. 3) is to represent such animation as a space-time complex made of key vertices and inbetween vertices (as defined previously), but also key edges and inbetween edges. A ***key (open) edge*** e_i is defined by a time t_i and a 2D curve $\phi_i(s)$, starting at a key vertex $v_{\text{start}}=(t_i,p_1)$ and ending at a key vertex $v_{\text{end}}=(t_i,p_2)$. An ***inbetween (open) edge*** $\mathbf{e}_j$ is defined by its *temporal boundary* and its *spatial boundary* (detailed in the next paragraph), from which can be computed a time-parameterized 2D curve $\Phi(s,t)$ (i.e., a surface in space-time) interpolating this boundary.



Naively (Fig. opposite), one might define the temporal boundary as a pair $(e_{\text{before}},e_{\text{after}})$ that references to two key edges to interpolate, and the spatial boundary as a pair $(\mathbf{v}_{\text{start}},\mathbf{v}_{\text{end}})$ that references to two inbetween vertices where the time-parameterized curve $\Phi(s,t)$ should start and end (for t fixed). Unfortunately, this naive definition would only enable to

represent a very small subset of all possible topological events that can happen to a stroke graph, and therefore we need a more general definition (Fig. 4, Left). Indeed, to represent an edge being cut in half by an appearing vertex (Fig. 3), or cut in more pieces by several vertices appearing simultaneously, we need the temporal boundary not to be two key edges, but two "sequences of connected key edges", structure that we call ***path***. To represent an edge growing from a vertex, we need to allow paths to be reduced to a key vertex. Finally, To allow *partial keyframing* (e.g., adding a key to an inbetween edge without adding a key to every incident edge, and recursively to every connected edge), we need the spatial boundary not to be two inbetween vertices, but two "sequences of connected inbetween vertices", structure that we call ***animated vertex*** (it is a chain key—inbetween—key—$\cdots$—key—inbetween—key, which can be interpreted as a vertex animated using conventional keyframing).

3.3 Animating vector graphics complexes

We extend these ideas further to represent an animated vector graphics complex [Dalstein et al. 2014] with time-varying topology. The same way that the VGC extends stroke graphs with closed edges and faces, the VAC extends the representation introduced in the previous section with key closed edges, inbetween closed edges, key faces, and inbetween faces.

A ***key closed edge*** e_i is defined by a time t_i and a 2D closed curve $\phi_i(s)$ (note that is does not have bounding vertices). An ***inbetween closed edge*** $\mathbf{e}_j$ is defined by its temporal boundary, made of two ***cycles*** (Fig. 4, Right), from which can be computed a time-parameterized 2D closed curve $\Phi(s,t)$ interpolating this boundary. Note that unlike inbetween open edges, inbetween closed edges have an empty spatial boundary, since closed edges do not have bounding vertices. To allow all sorts of topological events, cycles can either be reduced to a single key vertex, or made of a single (possibly repeated) key closed edge, or made of connected key open edges.

A ***key face*** f_i is defined by a time t_i and a sequence of cycles, all sharing the same time t_i. Given a winding rule (e.g., *even-odd* or *non-zero*), these cycles define a 2D region of the time-plane $t = t_i$. An ***inbetween face*** $\mathbf{f}_j$ is defined by its temporal boundary and its spatial boundary. Its temporal boundary is defined by two sequences of faces, the *before faces* and the *after faces*. Its spatial boundary is defined by a sequence of ***animated cycles***, structure that we introduce informally in the next three paragraphs.

Animated cycle We have seen that an *animated vertex* is a combinatorial structure that stores references to existing inbetween vertices, which define a time-parameterized position $p(t)$. Similarly, our goal is now to define a time-parameterized closed curve $\Phi(s,t)$, via a combinatorial structure storing references to existing cells (a "cylinder in space-time", cf. Fig. 5, Top-left). A simple option would be to define this boundary as a set $\{c_1, \ldots, c_n\}$ of references to cells. However, for the same reasons that this approach fails to define the boundary of key faces (cf. [Dalstein et al. 2014]), this approach fails to define the boundary of inbetween faces too. More specifically, because overlapping of cells is allowed (i.e., the complex is only *immersed* in space-time, as opposed to *embedded*), then the *set* of boundary cells does not contain enough information to unambiguously define the geometry of the face. Instead, it is necessary to *organize* this set using an ordered structure, possibly refering to the same cell multiple times. This additional information explicitly defines a *parameterization* of the boundary. For key faces, this is achieved via the structure called *cycle*. For inbetween faces, this is achieved via the structure called *animated cycle*.

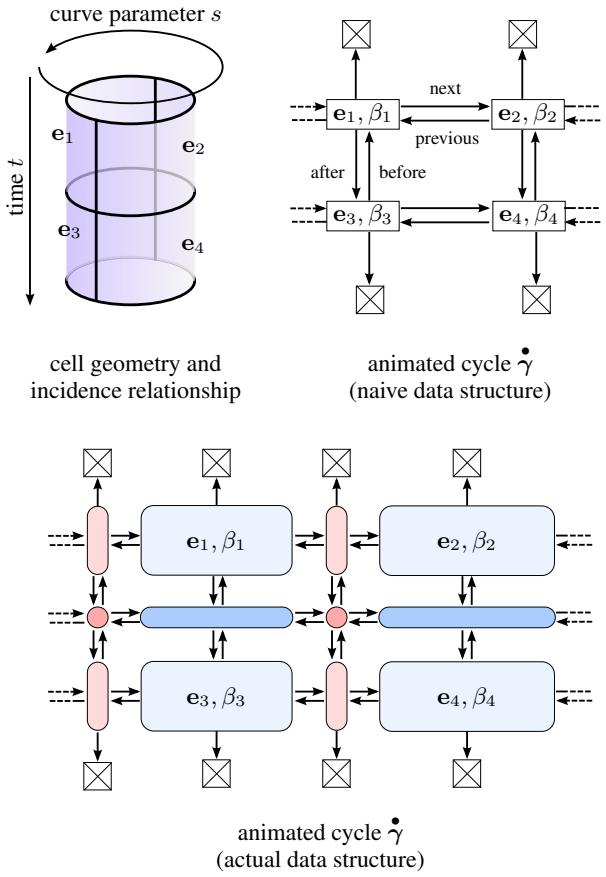


Figure 5: *Top: Intuitively, an animated cycle $\dot{\gamma}$ is a two-dimensional doubly linked list where every node holds a reference to an inbetween edge and an orientation. The structure is circular in the space dimension, and non-circular in the time dimension. Unfortunately, this naive structure is not expressive enough to capture all possible scenarios. Bottom: The actual data structure includes additional nodes to explicitly hold a reference to shared vertices, key edges, and inbetween vertices.*

Intuitively (Fig. 5, Top-right), a naive structure to define such parameterization would be a two-dimensional doubly linked list of oriented inbetween edges, where the first dimension corresponds to the curve parameter s, and the second dimension correspond to the time t. This structure is circular in the s-dimension, but non-circular in the t-dimension. Like a doubly linked list, it is composed of nodes which store: 1) per-node data; and 2) references to adjacent nodes. But unlike a doubly linked list, each node stores four references instead of two: ***previous*** and ***next*** to navigate in the s-dimension; and ***before*** and ***after*** to navigate in the t-dimension. The node data itself is a reference to an inbetween edge $\mathbf{e}$, and a boolean β that orients $\mathbf{e}$ with respect to curve parameterization.

Unfortunately, this naive structure cannot handle inbetween edges bounded by more than two key edges, more than two inbetween vertices, or that shrink to a key vertex, and thus cannot represent general time-parameterized cycles (e.g., Fig. 6, Left). Our solution is to include all the lower dimensional cells shared between inbetween edges as explicit nodes of the structure (Fig. 5, Bottom; Fig. 6, Right). It introduces a little redundancy to the structure, but makes it significantly more expressive.

4 Formal Definition

A ***vector animation complex*** $\mathcal{K}$ is defined as a tuple

$$\mathcal{K} = (C, \dim_T, \dim_S, \dots) \tag{1}$$

where C is a finite set of abstract symbols called ***cells*** (think of them as *identifiers*, or *addresses*), and $\dim_T, \dim_S, \dots$ are functions defined on C or a subset of C, assigning to relevant cells some ***attributes***, that have to satisfy some ***invariants***. These numerous attributes and invariants are detailed in the remainder of this section. In our C++ implementation, an element $c \in C$ is a *pointer* to an object inheriting the class `Cell`, and an attribute $\alpha(c)$ is typically a data member `c->m_alpha`.

Cell attributes can be classified in two types: ***topological attributes***, which are combinatorial objects defining incidence relationship between cells; and ***geometrical attributes***, which are continuous objects immersing the cells in space-time. The two most important attributes of any cell $c \in C$ are topological:

- its ***temporal dimension*** $\dim_T(c) \in \{0, 1\}$
- its ***spatial dimension*** $\dim_S(c) \in \{0, 1, 2\}$

Cells of temporal dimension 0 are called ***key cells***, and cells of temporal dimension 1 are called ***inbetween cells***. Orthogonally, cells of spatial dimension 0 are called ***vertices***, cells of spatial dimension 1 are called ***edges***, and cells of spatial dimension 2 are called ***faces***. In addition, each edge e is assigned the topological attribute $\text{isClosed}(e) \in \{\text{true}, \text{false}\}$. Therefore, the cells in C can be partitionned into eight finite sets which define their ***type***:

$\dim_T$	$\dim_S$	isClosed	Type	Notation
0	0	n/a	***key vertex***	$v \in V$
0	1	true	***key closed edge***	$e \in E_\circ$
0	1	false	***key open edge***	$e \in E_\mid$
0	2	n/a	***key face***	$f \in F$
1	0	n/a	***inbetween vertex***	$\mathbf{v} \in \mathbf{V}$
1	1	true	***inbetween closed edge***	$\mathbf{e} \in \mathbf{E}_\circ$
1	1	false	***inbetween open edge***	$\mathbf{e} \in \mathbf{E}_\mid$
1	2	n/a	***inbetween face***	$\mathbf{f} \in \mathbf{F}$

For convenience, we define $E = E_\mid \cup E_\circ$ and $\mathbf{E} = \mathbf{E}_\mid \cup \mathbf{E}_\circ$. In Section 4.1, we define all the remaining attributes and invariants for each type of cells. For clarity, some of these attributes are expressed using auxiliary structures (halfedges, paths, cycles, animated vertices, and animated cycles), which are defined in Section 4.2.

4.1 Cell attributes and invariants

Key vertex A key vertex $v \in V$ represents a single point in space-time:

topological attributes:	$\varnothing$	
geometrical attributes:	***position***	$p(v) \in \mathbb{R}^2$
	time	$t(v) \in \mathbb{R}$
invariants:	$\varnothing$	

Key closed edge A key closed edge $e \in E_\circ$ represents a closed curve contained in a time-plane:

topological attributes:	$\varnothing$	
geometrical attributes:	***curve***	$\phi(e) : s \in [0,1] \to \mathbb{R}^2$
	time	$t(e) \in \mathbb{R}$
invariants:	$\phi(e)$ continuous	
	$\phi(e)(0) = \phi(e)(1)$	

Key open edge A key open edge $e \in E_\mid$ represents an open curve contained in a time-plane, starting and ending at two key vertices (possibly equal):

topological attributes:	***start vertex***	$v_{\text{start}}(e) \in V$
	end vertex	$v_{\text{end}}(e) \in V$
geometrical attributes:	***curve***	$\phi(e) : s \in [0,1] \to \mathbb{R}^2$
	time	$t(e) \in \mathbb{R}$
invariants:	$\phi(e)$ continuous	
	$\phi(e)(0) = p(v_{\text{start}}(e))$	
	$\phi(e)(1) = p(v_{\text{end}}(e))$	
	$t(v_{\text{start}}(e)) = t(e) = t(v_{\text{end}}(e))$	

Key face A key face $f \in E$ represents a region of a time-plane delimited by closed curves (possibly self-intersecting, including going back and forth the same path or being reduced to a single point):

topological attr:	***cycles***	$\forall i \in [1..k(f)], \gamma_i(f) \in \Gamma$
		where $k(f) \geq 0$
geometrical attr:	***winding rule***	$R(f) \subseteq \mathbb{N}$
	time	$t(f) \in \mathbb{R}$
invariants:	$\forall i \in [1..k(f)], t(f) = t(\gamma_i(f))$	

Inbetween vertex An inbetween vertex $\mathbf{v} \in \mathbf{V}$ represents an interpolation in time between two key vertices:

topological at.:	***before vertex***	$v_{\text{before}}(\mathbf{v}) \in V$
	after vertex	$v_{\text{after}}(\mathbf{v}) \in V$
geometrical at.:	***animated position***	$\mathbf{p}(\mathbf{v}) : t \in [t_1, t_2] \to \mathbb{R}^2$
		where $t_1 = t(v_{\text{before}}(\mathbf{v}))$
		$t_2 = t(v_{\text{after}}(\mathbf{v}))$
invariants:	$t_1 < t_2$	
	$\mathbf{p}(\mathbf{v})$ continuous	
	$\mathbf{p}(\mathbf{v})(t_1) = p(v_{\text{before}}(\mathbf{v}))$	
	$\mathbf{p}(\mathbf{v})(t_2) = p(v_{\text{after}}(\mathbf{v}))$	

Inbetween closed edge An inbetween closed edge $\mathbf{e} \in \mathbf{E}_\circ$ represents an interpolation in time between two cycles:

t. at.:	***before cycle***	$\gamma_{\text{before}}(\mathbf{e}) \in \Gamma$
	after cycle	$\gamma_{\text{after}}(\mathbf{e}) \in \Gamma$
g. at.:	***animated curve***	$\Phi(\mathbf{e}) : (s,t) \in [0,1] \times [t_1, t_2] \to \mathbb{R}^2$
		where $t_1 = t(\gamma_{\text{before}}(\mathbf{e}))$
		$t_2 = t(\gamma_{\text{after}}(\mathbf{e}))$
inv.:	$t_1 < t_2$	
	$\Phi(\mathbf{e})$ continuous	
	$\forall t \in [t_1, t_2], \Phi(\mathbf{e})(0,t) = \Phi(\mathbf{e})(1,t)$	
	$\forall s \in [0,1], \Phi(\mathbf{e})(s,t_1) = \phi(\gamma_{\text{before}}(\mathbf{e}))(s)$	
	$\forall s \in [0,1], \Phi(\mathbf{e})(s,t_2) = \phi(\gamma_{\text{after}}(\mathbf{e}))(s)$	

Inbetween open edge An inbetween open edge $\mathbf{e} \in \mathbf{E}_\mid$ represents an interpolation in time between two paths, spatially bounded by two animated vertices:

t. at.:	***before path***	$\pi_{\text{before}}(\mathbf{e}) \in \Pi$
	after path	$\pi_{\text{after}}(\mathbf{e}) \in \Pi$
	start animated vertex	$\dot{\mathbf{v}}_{\text{start}}(\mathbf{e}) \in \dot{\mathbf{V}}$
	end animated vertex	$\dot{\mathbf{v}}_{\text{end}}(\mathbf{e}) \in \dot{\mathbf{V}}$

g. at.: ***animated curve*** $\Phi(\mathbf{e}) : (s,t) \in [0,1] \times [t_1, t_2] \to \mathbb{R}^2$
where $t_1 = t(\pi_{\text{before}}(\mathbf{e}))$, $t_2 = t(\pi_{\text{after}}(\mathbf{e}))$

inv.: $v_{\text{start}}(\pi_{\text{before}}(\mathbf{e})) = v_{\text{before}}(\overset{\bullet}{\mathbf{v}}_{\text{start}}(\mathbf{e}))$
$v_{\text{end}}(\pi_{\text{before}}(\mathbf{e})) = v_{\text{before}}(\overset{\bullet}{\mathbf{v}}_{\text{end}}(\mathbf{e}))$
$v_{\text{start}}(\pi_{\text{after}}(\mathbf{e})) = v_{\text{after}}(\overset{\bullet}{\mathbf{v}}_{\text{start}}(\mathbf{e}))$
$v_{\text{end}}(\pi_{\text{after}}(\mathbf{e})) = v_{\text{after}}(\overset{\bullet}{\mathbf{v}}_{\text{end}}(\mathbf{e}))$

$t_1 < t_2$

$\Phi(\mathbf{e})$ continuous
$\forall t \in [t_1, t_2], \Phi(\mathbf{e})(0,t) = \mathbf{p}(\overset{\bullet}{\mathbf{v}}_{\text{start}}(\mathbf{e}))(t)$
$\forall t \in [t_1, t_2], \Phi(\mathbf{e})(1,t) = \mathbf{p}(\overset{\bullet}{\mathbf{v}}_{\text{end}}(\mathbf{e}))(t)$

$\forall s \in [0,1], \Phi(\mathbf{e})(s,t_1) = \phi(\pi_{\text{before}}(\mathbf{e}))(s)$
$\forall s \in [0,1], \Phi(\mathbf{e})(s,t_2) = \phi(\pi_{\text{after}}(\mathbf{e}))(s)$

Inbetween face An inbetween face $\mathbf{f} \in \mathbf{F}$ represents an interpolation in time between key faces, spatially bounded by animated cycles:

top. at.: ***before time*** $t_{\text{before}}(\mathbf{f}) \in \mathbb{R}$
before faces $\forall i \in [1..k_b(\mathbf{f})], f_{\text{before},i}(\mathbf{f}) \in F$ where $k_b(\mathbf{f}) \geq 0$
after time $t_{\text{after}}(\mathbf{f}) \in \mathbb{R}$
after faces $\forall i \in [1..k_a(\mathbf{f})], f_{\text{after},i}(\mathbf{f}) \in F$ where $k_a(\mathbf{f}) \geq 0$
animated cycles $\forall i \in [1..k(\mathbf{f})], \overset{\bullet}{\boldsymbol{\gamma}}_i(\mathbf{f}) \in \overset{\bullet}{\boldsymbol{\Gamma}}$ where $k(\mathbf{f}) \geq 0$

geo. at.: ***winding rule*** $R(\mathbf{f}) \subseteq \mathbb{N}$

inv.: $\forall i \in [1..k_b(\mathbf{f})], \quad t_{\text{before}}(\mathbf{f}) = t(f_{\text{before},i}(\mathbf{f}))$
$\forall i \in [1..k_a(\mathbf{f})], \quad t_{\text{after}}(\mathbf{f}) = t(f_{\text{after},i}(\mathbf{f}))$
$\forall i \in [1..k(\mathbf{f})], \quad t_{\text{before}}(\mathbf{f}) = t_{\text{before}}(\overset{\bullet}{\boldsymbol{\gamma}}_i(\mathbf{f}))$
$\forall i \in [1..k(\mathbf{f})], \quad t_{\text{after}}(\mathbf{f}) = t_{\text{after}}(\overset{\bullet}{\boldsymbol{\gamma}}_i(\mathbf{f}))$

4.2 Auxiliary structures

Halfedge A ***halfedge*** is a pair $h = (e, \beta) \in E \times \{\top, \bot\}$. If e is closed then it is a ***closed halfedge*** denoted $h \in H_\circ$, otherwise it is an ***open halfedge*** denoted $h \in H_|$. If $\beta = \top$, we define $\phi(h)(s) = \phi(e)(s)$, otherwise we define $\phi(h)(s) = \phi(e)(1-s)$. If h is open then we define $v_{\text{start}}(h) = v_{\text{start}}(e)$ and $v_{\text{end}}(h) = v_{\text{end}}(e)$ (when $\beta = \top$), or $v_{\text{start}}(h) = v_{\text{end}}(e)$ and $v_{\text{end}}(h) = v_{\text{start}}(e)$ (when $\beta = \bot$). Finally, we define $t(h) = t(e)$.

Path A ***path*** π is either:

1. a key vertex $v \in V$, or
2. a list of $N > 0$ open halfedges $h_1, .., h_N \in H_|$ satisfying:

$$\forall j \in [1..N-1], \quad v_{\text{end}}(h_j) = v_{\text{start}}(h_{j+1})$$

In the first case, we define $v_{\text{start}}(\pi) = v_{\text{end}}(\pi) = v$, otherwise we define $v_{\text{start}}(\pi) = v_{\text{start}}(h_1)$ and $v_{\text{end}}(\pi) = v_{\text{end}}(h_N)$. Also, we define the curve $\phi(\pi) : s \in [0,1] \to \mathbb{R}^2$ by concatenating and uniformly reparameterizing the $\phi(h_j)$. In the special case $\pi = v$, then $\phi(\pi)$ is the constant function $\Phi(\pi)(s) = p(v)$. Finally, we define $t(\pi) = t(v)$ (Case 1.), or $t(\pi) = t(h_1)$ (Case 2.). We denote by Π the set of all possible paths on $\mathcal{K}$.

Cycle A ***cycle*** γ is either:

1. a key vertex $v \in V$, or
2. a closed halfedge $h \in H_\circ$ repeated $N > 0$ times, or
3. a circular list of $N > 0$ open halfedges $h_j \in H_|$ satisfying:

$$\forall j \in [1..N], \quad v_{\text{end}}(h_j) = v_{\text{start}}(h_{j+1})$$

In addition, a cycle stores a ***starting point*** $s_0 \in \mathbb{R}$. We define the closed curve $\phi(\gamma) : s \in [0,1] \to \mathbb{R}^2$ by concatenating and uniformly reparameterizing the $\phi(h_j)$, then offsetting by s_0. In the special case $\gamma = v$, then $\phi(\gamma)$ is the constant function $\Phi(\gamma)(s) = p(v)$. Finally, we define $t(\gamma) = t(v)$ (Case 1.), or $t(\gamma) = t(h)$ (Case 2.), or $t(\gamma) = t(h_1)$ (Case 3.). We denote by Γ the set of all possible cycles on $\mathcal{K}$.

Animated vertex An ***animated vertex*** $\overset{\bullet}{\mathbf{v}}$ is a list of $N > 0$ inbetween vertices $\mathbf{v}_1, .., \mathbf{v}_N \in \mathbf{V}$ satisfying:

$$\forall j \in [1..N-1], \quad v_{\text{after}}(\mathbf{v}_j) = v_{\text{before}}(\mathbf{v}_{j+1})$$

We define $v_{\text{before}}(\overset{\bullet}{\mathbf{v}}) = v_{\text{before}}(\mathbf{v}_1)$ and $v_{\text{after}}(\overset{\bullet}{\mathbf{v}}) = v_{\text{after}}(\mathbf{v}_N)$. Also, we define the time-parameterized position $\mathbf{p}(\overset{\bullet}{\mathbf{v}}) : t \in [t(v_{\text{before}}(\overset{\bullet}{\mathbf{v}})), t(v_{\text{after}}(\overset{\bullet}{\mathbf{v}}))] \to \mathbb{R}^2$ by concatenating the $\mathbf{p}(\mathbf{v}_j)$. We denote by $\overset{\bullet}{\mathbf{V}}$ the set of all possible animated vertices on $\mathcal{K}$.

Animated cycle An ***animated cycle*** is a tuple

$$\overset{\bullet}{\boldsymbol{\gamma}} = (N, c, \beta, n_{\text{previous}}, n_{\text{next}}, n_{\text{before}}, n_{\text{after}}) \tag{2}$$

where N is a non-empty set of symbols called ***nodes***, and:

$c : N \to V \cup \mathbf{V} \cup E \cup \mathbf{E}$		assigns a ***cell*** to every node
$\beta : N \to \{\top, \bot\}$		assigns an ***orientation*** (ignored if $c(n) \in V \cup \mathbf{V}$)
n_{previous} :	$N \to N$	assigns a ***previous*** node
n_{next} :	$N \to N$	assigns a ***next*** node
n_{before} :	$N \to N \cup \{\text{null}\}$	assigns an optional ***before*** node
n_{after} :	$N \to N \cup \{\text{null}\}$	assigns an optional ***after*** node

In addition, an animated cycle stores a ***starting node*** $n_0 \in N$. We define the ***timespan*** of a node n as being the trivial interval $T(n) = \{t(c(n))\}$ if $c(n)$ is a key cell, or the open interval $T(n) = (t_{\text{before}}(c(n)), t_{\text{after}}(c(n)))$ if $c(n)$ is an inbetween cell. Despite having a single *next* pointer, one can notice (Fig. 6) that when $c(n)$ is an inbetween open edge, then n may have several nodes "next to it", which are stacked in time. The *next* (resp. *previous*) pointer points to the "first" of these, and the others can be accessed by iterating *after* (resp. *before*). To easily traverse the data-structure at t fixed, we define the two functions $n_{\text{next}}(n,t)$ and $n_{\text{previous}}(n,t)$ that return the two nodes "spatially adjacent to n at time t".

$n_{\text{previous}}(n \in N, t \in \mathbb{R})$

Require: $t \in T(n)$
  $n' \leftarrow n_{\text{previous}}(n)$
  while $t \notin T(n')$ **do**
    $n' \leftarrow n_{\text{before}}(n')$
  return n'

$n_{\text{next}}(n \in N, t \in \mathbb{R})$

Require: $t \in T(n)$
  $n' \leftarrow n_{\text{next}}(n)$
  while $t \notin T(n')$ **do**
    $n' \leftarrow n_{\text{after}}(n')$
  return n'

We define the time-parameterized closed curve $\Phi(\overset{\bullet}{\boldsymbol{\gamma}})(s,t)$ by finding a node n such that $t \in T(n)$ (iterating before/after from n_0), then concatenating the $\phi(c(n))$ while iterating $n_{\text{next}}(n,t)$ (followed by a normalization into $[0,1]$). We denote by $\overset{\bullet}{\boldsymbol{\Gamma}}$ the set of all valid animated cycles on $\mathcal{K}$, which are the ones whose attributes satisfy the ***invariants*** that we provide in supplemental material, together with the definition of $t_{\text{before}}(\overset{\bullet}{\boldsymbol{\gamma}})$ and $t_{\text{after}}(\overset{\bullet}{\boldsymbol{\gamma}})$.

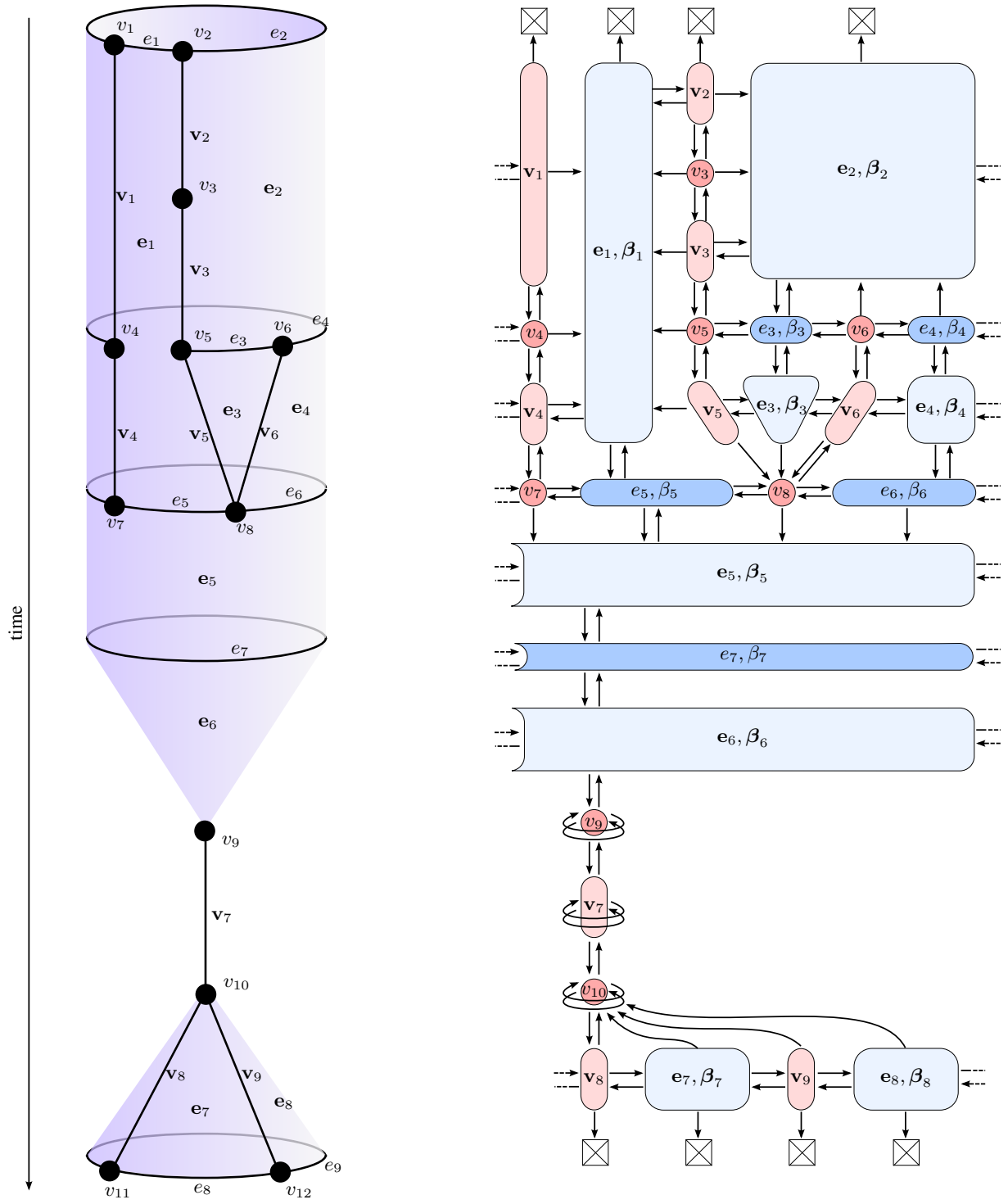
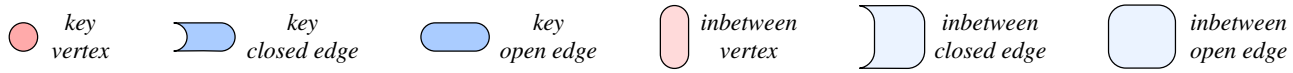


Figure 6: *A more general example of an animated cycle $\dot{\boldsymbol{\gamma}}$. Left: Geometry and topology of the cells $c \in C(\mathcal{K})$ involved in $\dot{\boldsymbol{\gamma}}$. It is a subcomplex of the whole VAC. Right: The nodes $n \in N(\dot{\boldsymbol{\gamma}})$ defining $\dot{\boldsymbol{\gamma}}$. Each node n references to a cell c, specifies an orientation β (ignored if c is a vertex), and points to a previous, next, before, and after node. The shape/color of the node indicates the type of the referenced cell:*



This example illustrates a variety of topological transformations over time for the cycle, including local keyframing using a key vertex (v_3), local keyframing using key edges (e_3, e_4), keyframing using a closed edge (e_7), contraction of an open edge to a vertex ($e_3 \to v_8$), contraction of a closed edge to a vertex ($e_7 \to v_9$), cutting an open edge into two open edges ($e_2 \to e_3, e_4$), among others.

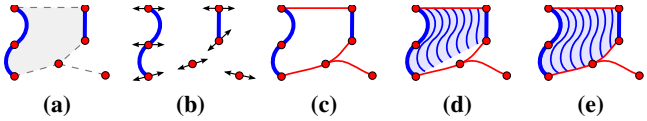


Figure 7: *Interpolation scheme (time = horizontal axis). (a) Input: geometry of key cells and space-time topology. (b) Compute tangents at key vertices. (c) Compute geometry of inbetween vertices, satisfying tangents. (d) For each inbetween edge, compute linear interpolation of bounding paths/cycles. (e) Output: warp to satisfy spatial boundary conditions.*

5 Interpolation Scheme

The geometry of inbetween cells may be provided explicitly (or in non-photorealistic rendering applications, computed from an animated 3D model), but in our case, it is computed by interpolating the geometry of key cells, as expected from a keyframing system.

First, for each key vertex v_i, we define a tangent $q(v_i)$ as the average of the slopes $\frac{p(v_j)-p(v_i)}{t(v_j)-t(v_i)}$, for all key vertices v_j connected to v_i by an inbetween vertex (Fig. 7b). Then, we define the geometry of each inbetween vertex as the unique cubic curve interpolating the positions $p(v_{\text{before}})$ and $p(v_{\text{after}})$ with the desired tangents $q(v_{\text{before}})$ and $q(v_{\text{after}})$ (Fig. 7c). All that is left to do is define the geometry of every inbetween open (resp. closed) edge, by interpolating its two bounding paths (resp. cycles). We recall from Section 4 that paths/cycles have an explicit parameterization $[0,1] \to \mathbb{R}^2$, obtained by concatenating and uniformly reparameterizing the key edges' parameterizations (the starting point of cycles is a user-controllable variable). First, we compute a linear interpolation between these two explicit parameterizations (Fig. 7d). Finally, in the case of inbetween open edges, for all $t \in (t_1, t_2)$, we linearly warp this interpolation $\Phi(s,t)$ such that $\Phi(0,t)$ and $\Phi(1,t)$ coincidate with the start and end animated vertices at t (Fig. 7e). There is no need to define an interpolation scheme for inbetween faces, since their geometry is entirely specified by the geometry of their boundary. Indeed, for all $t \in (t_1, t_2)$, a closed parameterized curve $[0,1] \to \mathbb{R}^2$ can be extracted from each animated cycle, which, together with the user-specified winding rule (e.g., even-odd), define an area of the 2D plane.

This interpolation scheme is robust and general but limited as it only guarantees $\mathcal{C}^0$ continuity. More aesthetically pleasing interpolation can be achieved using logarithmic spirals [Whited et al. 2010] or Coons patches. This is left for future work.

6 User Interface

To create and manipulate VACs, we implemented various visualizations and topological operators, which we present in this section. We refer to the accompanying video for a demonstration of these tools.

2D view We provide a 2D view to render a specific frame of the animation (i.e., a time-slice of the VAC), which can be selected using a timeline similar to any animation system. The 2D view can be split into multiple 2D views to visualize simultaneously different frames of the animation. The user can also toggle "onion skinning" to overlay several frames within a single 2D view, or render the animation as an animation strip (Fig.1, bottom). The frames can be rendered either in "normal" mode (showing the actual result), or in "topology" mode (using a color code to inform whether a cell is a key cell, or a time-slice of an inbetween cell).

3D view We also provide a 3D view to visualize the VAC in space-time. However, we mostly use this view as a debugging tool, as it becomes quickly impractical when the number of cells grow. All interaction happens in the 2D views, and all examples presented in this paper have been created without using the 3D view at all. At this stage, it is unclear whether it is relevant to expose such visualization to end users.

Creating key cells Key cells are created in the 2D view using standard VGC tools. They are assigned the time t_i selected in the timeline.

Motion-pasting The easiest way to create inbetween cells is to select key cells at time t_1, trigger the *copy* action, then move to time t_2 and trigger the *motion-paste* action. It creates a copy of the key cells, assigns them the time t_2, and creates inbetween cells that connect in time the old key cells to the new ones. In other words, it corresponds to sweeping key cells in time. Once motion-pasted, the new cells can be edited to create the desired motion. Standard VGC topological operators (extended to support incident inbetween cells) can also be used on the new key cells, which introduce topological events as a result.

Inbetweening Another way to create inbetween cells is to select existing key cells at two different times t_1 and t_2 (e.g., using side-by-side 2D views), then trigger the *inbetweening* action. It creates inbetween cells that connect in time the selected key cells. Currently, it works to create an inbetween vertex out of two key vertices; an inbetween edge out of two key edges; an inbetween edge out of more than two key edges that can be organized into two paths or two cycles; or an inbetween edge that grows or shrinks to a vertex. This tool does not yet support the creation of inbetween faces (we can still create them using motion-pasting or manually specifying their boundary), neither the simultaneous creation of multiple inbetween edges, which are both interesting challenges left as future work.

Inserting keys A fundamental topological operator on VACs is to cut an inbetween cell in half, in the time dimension, by inserting a new key cell. It is the equivalent of inserting a keyframe in conventional keyframing animation. Similarly to the "auto-key" feature of most animation systems, we automatically call this operator whenever the user performs an action on (the rendered time-slice of) an inbetween cell. For instance, attempting to move an inbetween vertex automatically inserts a key vertex at the time t_i selected in the timeline, and the new key vertex is the cell actually moved. Note that this *insert key* tool is local: it cuts the selected inbetween cell and its spatial boundary, but does not propagate to any other cell. This allows for local trajectory or topology refininement possible, without keyframing the whole drawing.

Drag-and-drop Selected key cells can be drag-and-dropped in space (using the 2D view), but also in time (using the timeline), within a time interval determined by its incident inbetween cells. This allows to easily refine the timing of a motion.

Depth-ordering We store a global ordering for all the cells in a complex using a doubly-linked list, and render the cells back-to-front using this ordering. As with VGCs, we provide tools to conveniently alter this ordering.

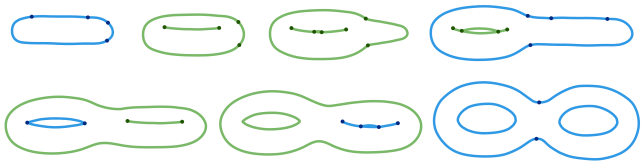


Figure 8: *Double Torus.*

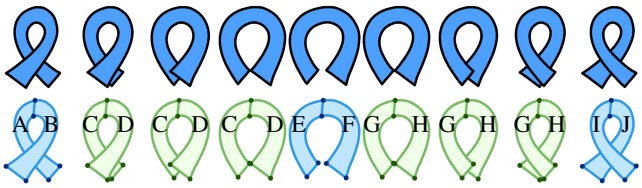


Figure 9: *Animated ribbon decomposed into 6 key faces (A,B,E,F,I,J) and 4 inbetween faces (C,D,G,H), in order to depict local depth-ordering both in space and time.*

7 Results

We create several illustrative examples of vector graphics animations that involve topological changes over time. We briefly summarize them below, although they are best seen in the video that accompanies this paper.

Torus The torus (Fig. 1) is an example use of the VAC to create a clean conceptual animated vector graphics illustration. It is defined using a total of five *keyframes* (frames that contain at least one key cell). As always required, the clip begins and ends with *full keyframes* (frames containing key cells only), which specify the shape of all the drawing elements that exist at those key times. The second keyframe captures the motion of the interior silhouette, marks the initial appearance of the hole with a single vertex, and also keyframes the shape of the outside silhouette. The third keyframe properly introduces the now-visible hole, while the fourth keyframe then ends the growth of the hole by merging the end vertices of the two lines. As seen in this example, keyframes are used either to introduce a change in shape, to introduce a change in topology, or both. Keyframes are typically *local*, i.e., key cells are only inserted where needed, without keyframing the entire drawing.

Double torus Once we have the VAC for the single torus, the animation of a double torus (Fig. 8) is easy to create. Indeed, all that is needed is: 1) deforming the outside silhouette, 2) copy-pasting to a different space-time location the sub-complex representing the animation of the hole, and 3) gluing the first key edge of this sub-complex to the deformed silhouette. We believe that this type of template-based construction provides a practical way of simplifying the creation of VACs. Figure 8 shows a vector graphics animation of a simple torus which is morphed to a double torus, with the two halves rotating asynchronously. Creating such animation would be hard using conventional vector graphics tools, but would be equally hard in 3D, since the genus of the depicted surface changes, requiring a topological event in 3D as well.

Animated ribbon In a given VAC, any cell is either completely in front, or completely behind, any other cell. However, any cell can be easily split spatially (cutting) or temporally (keyframing) into different cells, and the cells of this new cell-decomposition are assigned their own independent depth orders. This makes possible

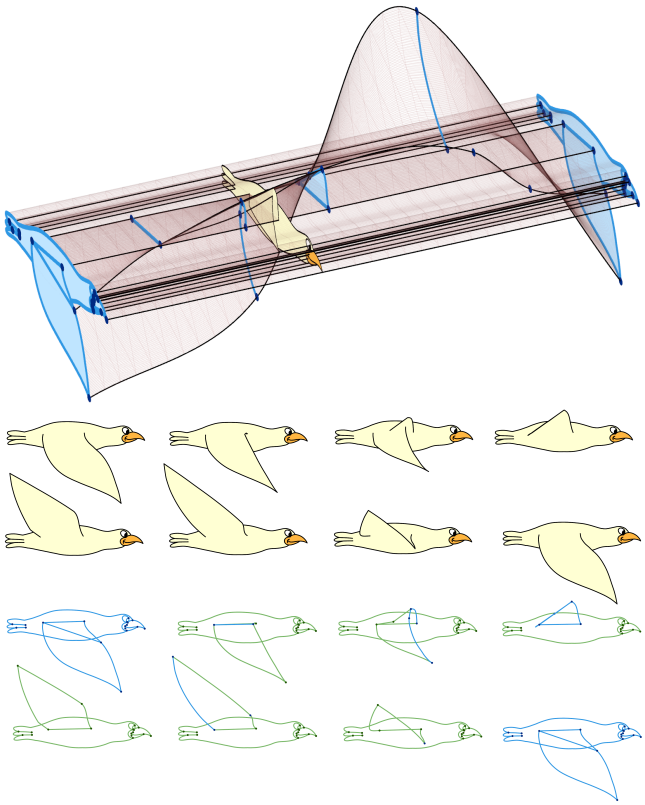


Figure 10: *Bird animation. Space-time view (top); output animation (middle); VGC film strip (bottom).*

to depict local depth-ordering, both in space and time, as illustrated in Figure 9. Using motion-pasting and basic editing, the space-time topology and geometry of this animation can be created within a few seconds. Then, the user alters the depth-ordering to ensure $A < B, C < D, G > H$, and $I > J$, using the "raise" and "lower" actions, as with standard VGCs. Note that this example does not contain topological events: keyframes are only used to introduce a change in geometry, as well as a change in depth-ordering.

Flapping bird We demonstrate Figure 10 the use of animated faces with depth layering in creating an example of a bird with a flapping wing, as inspired by a hand-drawn animation [Blair 1994, p. 122]. This example is created using 7 keyframes, all of which are local except the ones at the start and end. A looping animation is easily created by copy-and-pasting a second copy of the full VAC so that it sits immediately after the first copy. The ending elements of the first animation are then topologically glued to the starting elements of the second animation.

Head turning We use a drawn animation sequence by James Lopez (used with permission) as inspiration for a more complex example, shown in Figure 11. This involves many drawn elements and a significant number of topological changes, particularly involving the ear, goggles, eye, and mouth. Many topological changes need not be modeled in great detail. The eye is a good example: the features of the eye are simply spawned from an initial vertex that is introduced on the silhouette of the face. For this example, the 3D space-time view is largely unusable because of its complexity, and thus it proved to be a good test case for the capabilities of our user interface.

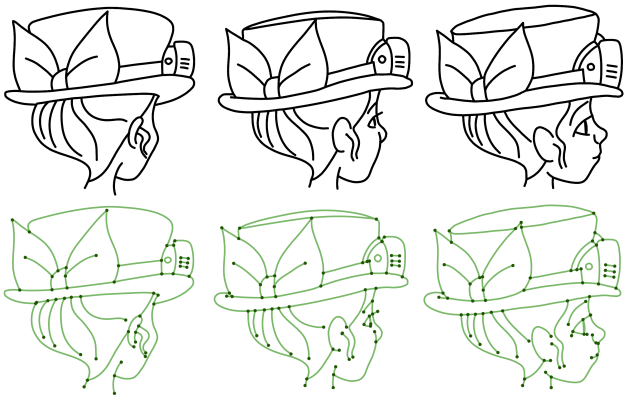


Figure 11: *Turning head animation. Output animation (top); VGC film strip (bottom).*

8 Discussion

Creation Many aspects of working with the VAC are no different than that of creating a conventional keyframe animation. Animation workflows are often classified as being *straight ahead* or *pose-to-pose*, and these working styles can each be reproduced using the VAC. A *straight ahead* workflow is readily reproduced using motion-pasting to create a new keyframe, followed by editing as necessary. A *pose-to-pose* workflow can be modeled by creating independent keyframes, followed by the creation of inbetween cells interpolating the key cells. For other potential applications, such as the vectorization of existing animations or video clips, we expect that the creation of the VAC may be automated.

Editing Creating the space-time topology of a complex animation may take more time than via traditional animation but once created, the VAC offers significant benefits as it provides a compact representation that is continuous in space and time. The VAC can be easily edited in ways that are not possible with traditional 2D or 3D animation pipelines. The VAC also provides a compact and convenient representation for algorithms to operate on. For example, we envision algorithms that can produce rich variations of an existing animated drawing by adding stochastic perturbations in space and time to some of the key elements.

Local keyframing Conventional keyframing animation allows for independent keyframing of the animation variables, i.e., the keyframe times for an animated knee-joint motion can be different from the keyframe times for the animated ankle motion within the same animation. Similarly, the VAC allows for the asynchronous specification of keyframes for portions of the vector graphics complex. Local keyframes provide better support for the semantics of many vector graphics drawings by allowing different portions of a drawing to be governed by different keyframes. It also allows for many topological changes to be conveniently modeled using instantiated templates.

Repurposing of exising 3D complexes It is tempting to believe that modeling an animated 2D complex could be achieved using existing approaches for 3D topological modeling, where the z-coordinate simply plays the role of time. Unfortunately, this does not reflect the unique semantics of the time axis, and this manifests itself in several ways. An "out of plane" rotation of a vector graphics animation does not usually produce a valid animation because the space is not Euclidean. For similar reasons, others have proposed representing image spaces as a non-Euclidean, Cayley-Klein geometry with one isotropic dimension [Koenderink and Doorn 2002]. Without a special designation for time, specific strategies would be needed to model the changing depth-orderings that can be desired during the course of a vector graphics animation, and which, by contrast, are easily modeled using the VAC. More importantly, cells would not always admit a time-parameterization. By contrast, all cells in our complex have an explicit time-parameterization, by design. This makes extracting time-slices trivial and also guarantees that all topological events are constrained to occur at key cells. This would not be the case if our cells were allowed to do "switch-backs" in time. In addition, despite being both 1D in space-time, the distinction we make between key edges and inbetween vertices is critical since their intersection with a time-plane is an object of different dimension. They must thus be rendered differently and store different attributes. The same is true for key faces and inbetween edges. Also, we allow zero-length edges but not zero-duration inbetween cells, i.e., we enforce $t_1 < t_2$. Similarly, paths are allowed to be reduced to a single key vertex, while animated vertices are not.

Using a simplicial complex representation for vector graphics animation [Southern 2008] would necessitate the use of many cells, which could then be problematic for creation, editing, and visualization, as well as being further removed from the standard keyframing paradigm for animation. Given a simplicial complex that completely reflects the geometry of an VAC, the VAC can be seen as inducing a partition of the simplicial complex, resulting in an output semantics similar to [Buchholz et al. 2011]. In general, geometric complexes allow for models and operations that we wish to forbid in order to reflect the unique nature of the time dimension. Implementing the desired constraints necessitates additional complexity while the VAC implements the desired constraints by design, i.e., as part of its *desiderata*. Also, we note that the intersection of a 3D simplicial complex with a time-plane is not necessarily a 2D simplicial complex (as the intersection between a tetrahedron and a plane can be a four-sided polygon). By contrast, the intersection of a VAC with a time-plane is guaranteed by design to be a VGC, which is trivial to compute due to the explicit time-parameterization.

Limitations While there are many benefits to a structure that provides a sound, continuous-time model of the topological events in vector graphics animations, it also comes with additional complexity. In particular, the modeling and editing of *animated cycles*, as required in order to animate faces in the vector graphics complex, embodies much of the complexity of the data structure and its implementation. The space-time topology is also likely to introduce a steep learning curve for artists coming from the world of SVG models where changes in topology are approximated by other means. We currently leave the development of an improved user interface as future work, and as such we have not conducted a formal user study with regard to how end users can best work with the VAC. We believe that the use of topological-event templates may significantly simplify the workflow for modeling and editing. Finally, our system shares the same fundamental limitation of any 2D animation system: the loss of information between the depicted 3D world and the 2D depiction [Catmull 1978]. In other words, the semantics of a rotating 3D object will always be better captured by 3D representations. We believe that the automatic computation of a VAC from an animated 3D model would alleviate this issue.

Future work The VAC opens up a number of exciting avenues for future work. Computing aesthetically pleasing interpolation between key cells is a rich and interesting problem. In conventional animation systems, animation curves are defined for any animation

variable by keyframes that always have well-defined *before* and *after* keyframes. This allows for well-defined tangent vectors to be specified or inferred at keyframes (e.g., Catmull-Rom). However, the topological events allowed by the VAC means that a key cell can have multiple *before* and *after* key cells, e.g., two or more vertices that join or split at a given time t, or an entire edge or face that merges or spawns from a given vertex. Developing sound and practical methods for position interpolation or user-based tangency specification is significantly more complex as a result.

Future work is needed to provide high-level manipulation of the VAC. For instance, a space-time paint bucket tool would be useful for creating inbetween faces. The automatic computation of inbetween cells from a general selection of key cells (i.e., automatic inbetweening) is a largely open problem, and extending [Whited et al. 2010] to the VAC is an exciting direction to explore. Also, we have developed a number of visualization tools in support of end user understanding, but much more is possible.

The topological structures could be further extended to allow the creation of motion graphs (equivalently, "move trees"), as is commonly done within game engines for character animation. This would require the ability to follow a given time-indexed "branch" of the VAC, and to rejoin existing branches. The ability to do this with local parts of a VGC would provide even further flexibility, although the resulting complexity might be difficult to develop and debug. One could also imagine creating additional continuous dimensions, such as that created by an aspect graph, i.e., creating a model that is parameterized with respect to the viewing direction as well as time.

An interesting direction is to develop VACs directly from video or rendering of a 3D model. VACs could be used to achieve continuous *space-time tracking*, as a logical extension of keyframe tracking for rotoscoping applications [Agarwala et al. 2004]. Interesting initial steps towards the vectorization of video have recently been explored [Patterson et al. 2012]. The data structure also has potential applications in non-photorealistic rendering, where there is a need for sound time-coherent models of the regions and strokes in an image sequence [Bénard et al. 2014]. Given the separation of topological and geometrical information in the VGC and the VAC, it should also be possible to develop a limited class of 3D animation using the VAC. Both of the above problems point to the need to develop good models for developing or otherwise modeling consistent parameterizations for edges and faces.

Some features supported by traditional vector graphics animation tools are not yet implemented, such as grouping, path-following, clipping, and masking. It is not yet clear how orthogonal this feature set is to the topological modeling aspects that we have focused on. Finally, there are interesting future directions to improve rendering and performance across the wide range of platforms that are a driving force behind increasing popularity of vector graphics.

9 Conclusions

We have presented a new data structure for representing vector graphics animation: the vector animation complex (VAC). It provides a compact, continuous-time continuous-space representation for vector graphics that is designed to support topological events. We expect that such continuous representations will become increasingly important as content needs to be developed for an ever-wider range of display resolutions and frame rates. Compared to conventional representations for vector graphics animation, the VAC captures more faithfully the semantics of many animations, therefore provides better support for manual editing or algorithm processing. Local keyframing is supported, i.e., keyframes need only provide information about the topological or shape changes for the subset of parts that require a given change. Topological changes can be modeled where they are desired and can be avoided where they are more simply modeled using other means, such as depth layering.

We envision that the VAC may be used in a wide range of applications, including the traditional domains for vector graphics animations; traditional drawing-based 2D animation, and the image-processing pipelines that are part of video processing and non-photorealistic rendering applications.

10 Acknowledgements

Special thanks go to James Lopez for allowing us to use his work from Hullabaloo, and his feedback on our work. We also thank all the reviewers for their helpful comments to improve the paper. Part of this work was supported by ERC Advanced Grant "Expressive", by GRAND, and by NSERC.

References

AGARWALA, A., HERTZMANN, A., SALESIN, D. H., AND SEITZ, S. M. 2004. Keyframe-based tracking for rotoscoping and animation. *ACM Trans. Graph. 23*, 3, 584–591.

ALEXA, M., COHEN-OR, D., AND LEVIN, D. 2000. As-rigid-as-possible shape interpolation. In *Proceedings of SIGGRAPH 2000*, 157–164.

ASENTE, P., SCHUSTER, M., AND PETTIT, T. 2007. Dynamic planar map illustration. *ACM Trans. Graph. 26*, 3, 30:1–30:10.

BAUDELAIRE, P., AND GANGNET, M. 1989. Planar maps: An interaction paradigm for graphic design. In *Proceedings of CHI '89*, 313–318.

BAXTER, W., BARLA, P., AND ANJYO, K.-I. 2009. Compatible embedding for 2d shape animation. *IEEE Trans. on Visualization and Computer Graphics 15*, 5, 867–879.

BÉNARD, P., LU, J., COLE, F., FINKELSTEIN, A., AND THOLLOT, J. 2012. Active strokes: Coherent line stylization for animated 3d models. In *Proceedings of NPAR '12*, 37–46.

BÉNARD, P., HERTZMANN, A., AND KASS, M. 2014. Computing smooth surface contours with accurate topology. *ACM Trans. Graph. 33*, 2, 19:1–19:21.

BLAIR, P. 1994. *Cartoon animation*. How to Draw and Paint Series. W. Foster Pub.

BREGLER, C., LOEB, L., CHUANG, E., AND DESHPANDE, H. 2002. Turning to the masters: Motion capturing cartoons. *ACM Trans. Graph. 21*, 3, 399–407.

BUCHHOLZ, B., FARAJ, N., PARIS, S., EISEMANN, E., AND BOUBEKEUR, T. 2011. Spatio-temporal analysis for parameterizing animated lines. In *Proceedings of NPAR '11*, 85–92.

BURTNYK, N., AND WEIN, M. 1971. Computer-generated key-frame animation. *Journal of the Society of Motion Picture & Television Engineers 80*, 3, 149–153.

CATMULL, E. 1978. The problems of computer-assisted animation. *SIGGRAPH Comput. Graph. 12*, 3, 348–353.

DALSTEIN, B., RONFARD, R., AND VAN DE PANNE, M. 2014. Vector graphics complexes. *ACM Trans. Graph. 33*, 4, 133:1–133:12.

DE FLORIANI, L., HUI, A., PANOZZO, D., AND CANINO, D. 2010. A dimension-independent data structure for simplicial complexes. In *Proceedings of the 19th International Meshing Roundtable*, 403–420.

DE JUAN, C. N., AND BODENHEIMER, B. 2006. Re-using traditional animation: methods for semi-automatic segmentation and inbetweening. In *Proceedings of SCA '06*, 223–232.

EISEMANN, E., PARIS, S., AND DURAND, F. 2009. A visibility algorithm for converting 3D meshes into editable 2D vector graphics. *ACM Trans. Graph. 28*, 3, 83:1–83:8.

FAUSETT, E., PASKO, A., AND ADZHIEV, V. 2000. Space-time and higher dimensional modeling for animation. In *Proceedings of Computer Animation 2000*, 140–145.

FEKETE, J.-D., BIZOUARN, E., COURNARIE, E., GALAS, T., AND TAILLEFER, F. 1995. TicTacToon: A paperless system for professional 2D animation. In *Proceedings of SIGGRAPH 95*, 79–90.

FIORE, F. D., SCHAEKEN, P., ELENS, K., AND REETH, F. V. 2001. Automatic in-betweening in computer assisted animation by exploiting 2.5D modelling techniques. In *Proceedings of Computer Animation 2001*, 192–200.

FU, H., TAI, C.-L., AND AU, O. K.-C. 2005. Morphing with laplacian coordinates and spatial-temporal texture. In *Proceedings of Pacific Graphics 2005*, 100–102.

IGARASHI, T., MOSCOVICH, T., AND HUGHES, J. F. 2005. As-rigid-as-possible shape manipulation. *ACM Trans. Graph. 24*, 3, 1134–1141.

KARSCH, K., AND HART, J. C. 2011. Snaxels on a plane. In *Proceedings of NPAR '11*, 35–42.

KOENDERINK, J. J., AND DOORN, A. J. V. 2002. Image processing done right. In *Proceedings of the 7th European Conference on Computer Vision*, 158–172.

KORT, A. 2002. Computer aided inbetweening. In *Proceedings of NPAR '02*, 125–132.

KWARTA, V., AND ROSSIGNAC, J. 2002. Space-time surface simplification and edgebreaker compression for 2D cel animations. *International Journal of Shape Modeling 8*, 2, 119–137.

LASSETER, J. 1987. Principles of traditional animation applied to 3d computer animation. *SIGGRAPH Comput. Graph. 21*, 4, 35–44.

LIENHARDT, P. 1994. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *International Journal of Computational Geometry & Applications 04*, 03, 275–324.

LIU, D., CHEN, Q., YU, J., GU, H., TAO, D., AND SEAH, H. S. 2011. Stroke correspondence construction using manifold learning. *Computer Graphics Forum 30*, 8, 2194–2207.

MCCANN, J., AND POLLARD, N. 2009. Local layering. *ACM Trans. Graph. 28*, 3, 84:1–84:7.

NGO, T., CUTRELL, D., DANA, J., DONALD, B., LOEB, L., AND ZHU, S. 2000. Accessible animation and customizable graphics via simplicial configuration modeling. In *Proceedings of SIGGRAPH 2000*, 403–410.

NORIS, G., SÝKORA, D., COROS, S., WHITED, B., SIMMONS, M., HORNUNG, A., GROSS, M., AND SUMNER, R. W. 2011. Temporal noise control for sketchy animation. In *Proceedings of NPAR '11*, 93–98.

NORIS, G., HORNUNG, A., SUMNER, R. W., SIMMONS, M., AND GROSS, M. 2013. Topology-driven vectorization of clean line drawings. *ACM Trans. Graph. 32*, 1, 4:1–4:11.

ORZAN, A., BOUSSEAU, A., WINNEMÖLLER, H., BARLA, P., THOLLOT, J., AND SALESIN, D. 2008. Diffusion curves: A vector representation for smooth-shaded images. *ACM Trans. Graph. 27*, 3, 92:1–92:8.

PATTERSON, J. W., TAYLOR, C. D., AND WILLIS, P. J. 2012. Constructing and rendering vectorised photographic images. *The Journal of Virtual Reality and Broadcasting 9*, 3.

PESCO, S., TAVARES, G., AND LOPES, H. 2004. A stratification approach for modeling two-dimensional cell complexes. *Computers & Graphics 28*, 2, 235–247.

RAVEENDRAN, K., WOJTAN, C., THUEREY, N., AND TURK, G. 2014. Blending liquids. *ACM Trans. Graph. 33*, 4, 137:1–137:10.

REEVES, W. T. 1981. Inbetweening for computer animation utilizing moving point constraints. *SIGGRAPH Comput. Graph. 15*, 3, 263–269.

RIVERS, A., IGARASHI, T., AND DURAND, F. 2010. 2.5D cartoon models. *ACM Trans. Graph. 29*, 4, 59:1–59:7.

ROSSIGNAC, J., AND O'CONNOR, M. 1989. *SGC: A Dimension-independent Model for Pointsets with Internal Structures and Incomplete Boundaries*. Research report. IBM T.J. Watson Research Center.

SEBASTIAN, T. B., KLEIN, P. N., AND KIMIA, B. B. 2003. On aligning curves. *IEEE Trans. on Pattern Analysis and Machine Intelligence 25*, 1, 116–125.

SEDERBERG, T. W., GAO, P., WANG, G., AND MU, H. 1993. 2-D shape blending: an intrinsic solution to the vertex path problem. In *Proceedings of SIGGRAPH 93*, 15–18.

SOUTHERN, R. 2008. Animation manifolds for representing topological alteration. Tech. Rep. UCAM-CL-TR-723, University of Cambridge, Computer Laboratory.

SÝKORA, D., DINGLIANA, J., AND COLLINS, S. 2009. As-rigid-as-possible image registration for hand-drawn cartoon animations. In *Proceedings of NPAR '09*, 25–33.

SÝKORA, D., BEN-CHEN, M., ČADÍK, M., WHITED, B., AND SIMMONS, M. 2011. TexToons: practical texture mapping for hand-drawn cartoon animations. In *Proceedings of NPAR '11*, 75–84.

THOMAS, F., AND JOHNSTON, O. 1987. *Disney Animation: The Illusion of Life*. Abbeville Press.

WEILER, K. 1985. Edge-based data structures for solid modeling in curved-surface environments. *IEEE Computer Graphics and Applications 5*, 1, 21–40.

WHITED, B., NORIS, G., SIMMONS, M., SUMNER, R., GROSS, M., AND ROSSIGNAC, J. 2010. BetweenIT: An interactive tool for tight inbetweening. *Computer Graphics Forum 29*, 2, 605–614.

YU, J., BIAN, W., SONG, M., CHENG, J., AND TAO, D. 2012. Graph based transductive learning for cartoon correspondence construction. *Neurocomputing 79*, 0, 105–114.

ZHANG, S.-H., CHEN, T., ZHANG, Y.-F., HU, S.-M., AND MARTIN, R. R. 2009. Vectorizing cartoon animations. *IEEE Trans. on Visualization and Computer Graphics 15*, 4, 618–629.