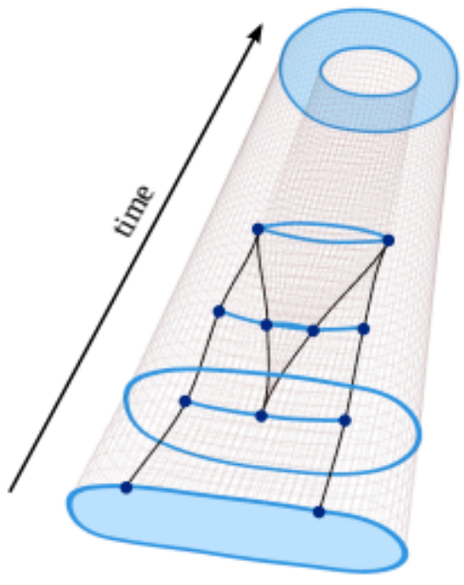
# READING:

## 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




Space-time visualization

| | key vertex | key closed edge | key open edge | key face | inbetween vertex | inbetween closed edge | inbetween open edge | inbetween face |
|---|---|---|---|---|---|---|---|---|
| number | 11 | 3 | 10 | 2 | 10 | 3 | 9 | 1 |
| space-time visualization | | | | | | | | ∅ |
| time-slices visualization | | | | | | | | |

**Legend**

Time-slices visualization

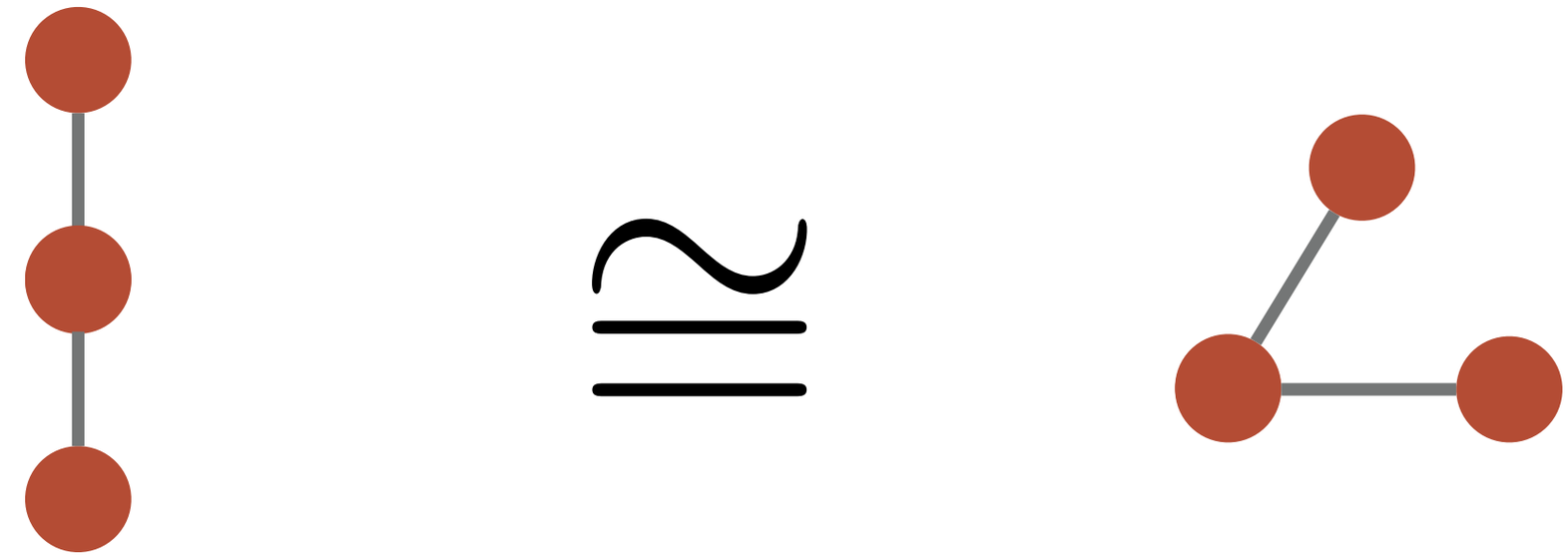
**Figure 1:** *A space-time continuous 2D animation depicting a rotating torus, created without 3D tools. First, the animator draws key cells (in blue) using 2D vector graphics tools. Then, he specifies how to interpolate them using inbetween cells (in green). Our contribution is a novel data structure,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

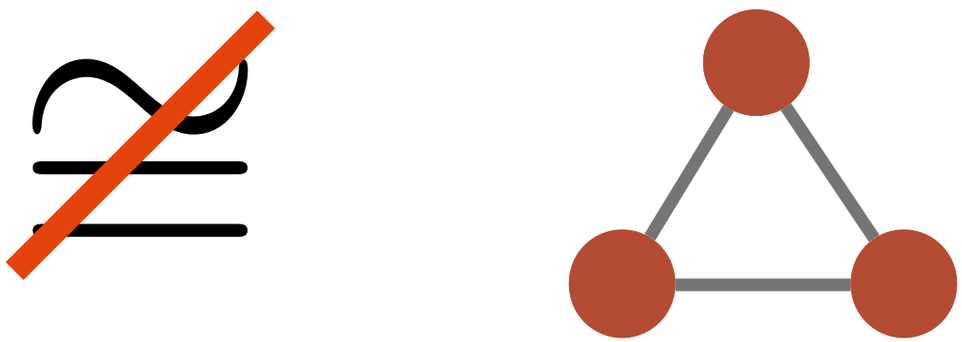
*Presenter:*
*Chenxi Liu*

# Topology in simple language

**A field about incidence relationships.**

# Topology in simple language

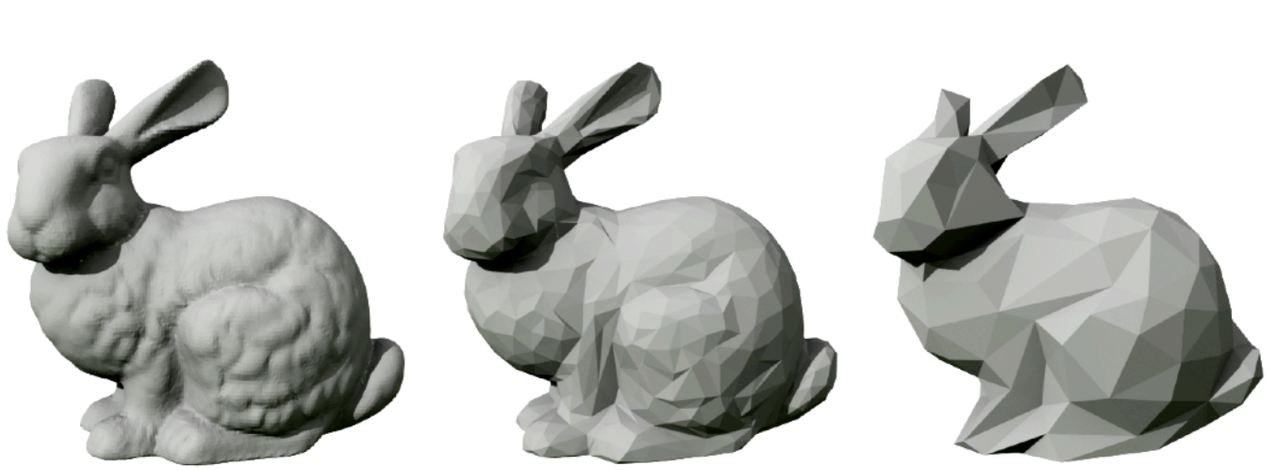




# Topology in simple language

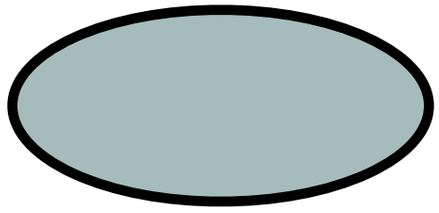
*https://www.youtube.com/watch?time_continue=2&v=9NlqYr6-TpA*

# Topology in 3D geometry processing



$\simeq$

$\simeq$



# Exceptions

## Topology-controlled Reconstruction of Multi-labelled Domains from Cross-sections

ZHIYANG HUANG, Washington University in St. Louis
MING ZOU, Washington University in St. Louis
NATHAN CARR, Adobe Systems
TAO JU, Washington University in St. Louis

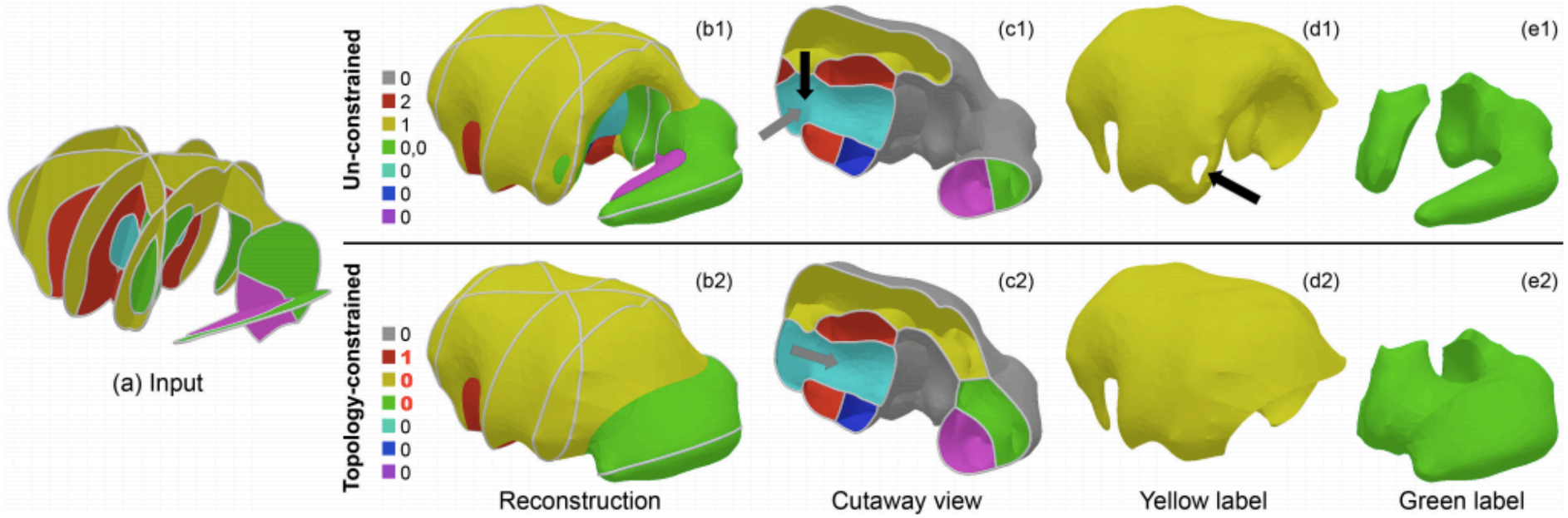
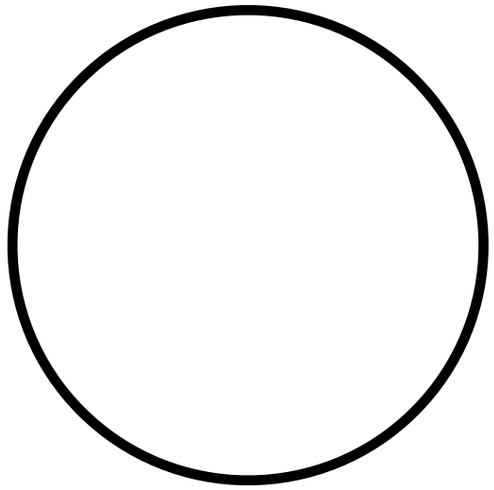
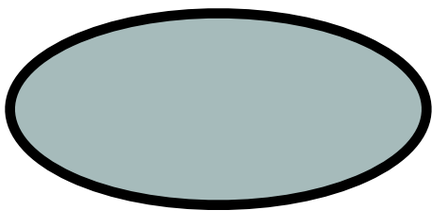



Fig. 1. Given several multi-labeled planes depicting the anatomical regions of a mouse brain (a), reconstruction without topology control (b1) leads to redundant handles for the red and yellow labels (black arrows in c1, d1) and disconnection for the green label (e1). Our method (b2) allows the user to prescribe the topology such that the red label has one tunnel (gray arrow in c2), the yellow label has no tunnels (d2), and the green label is connected (e2). The legends in (b1,b2) report, for each label in the reconstruction, the genus of each surface component bounding that label (e.g., "0,0" means two surfaces each with genus 0). User-specified constraints are colored red.

# 2D Graphics



## Lower dimension?

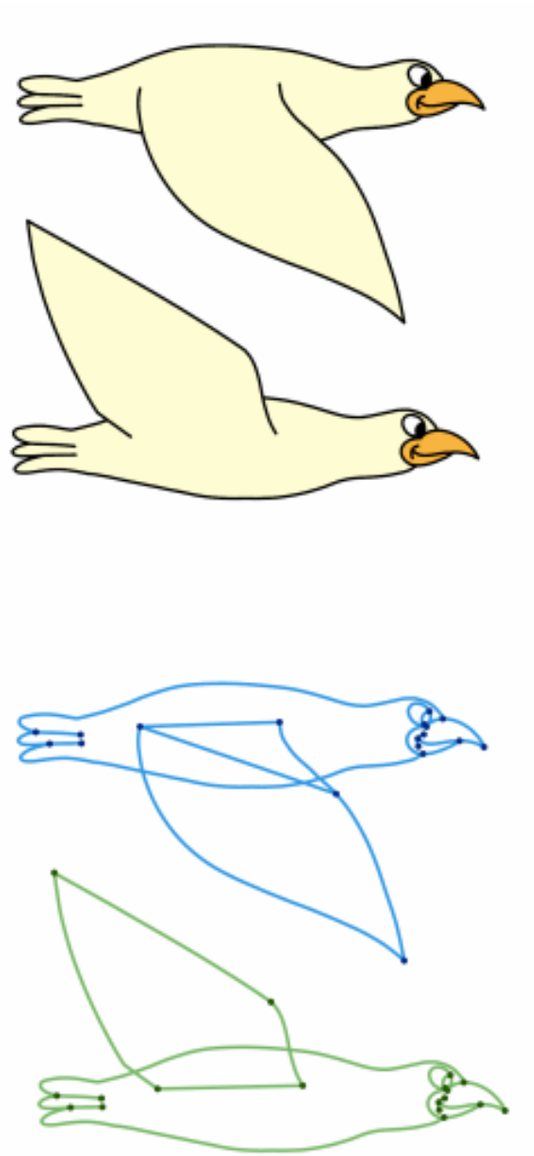


# 2D Graphics

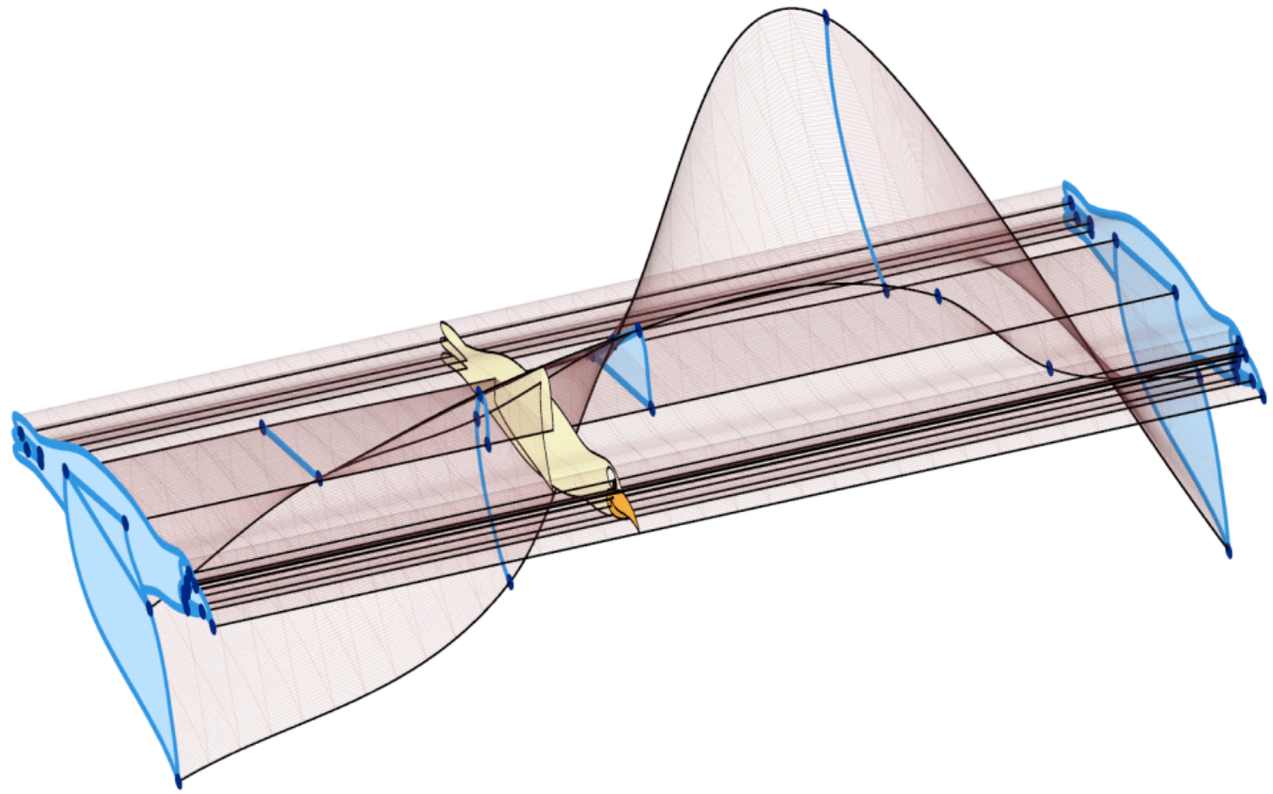
Lower dimension?

*https://media0.giphy.com/media/xT1TTHe611n64f5l1C/giphy.gif*

# Topological Modeling for Vector Graphics

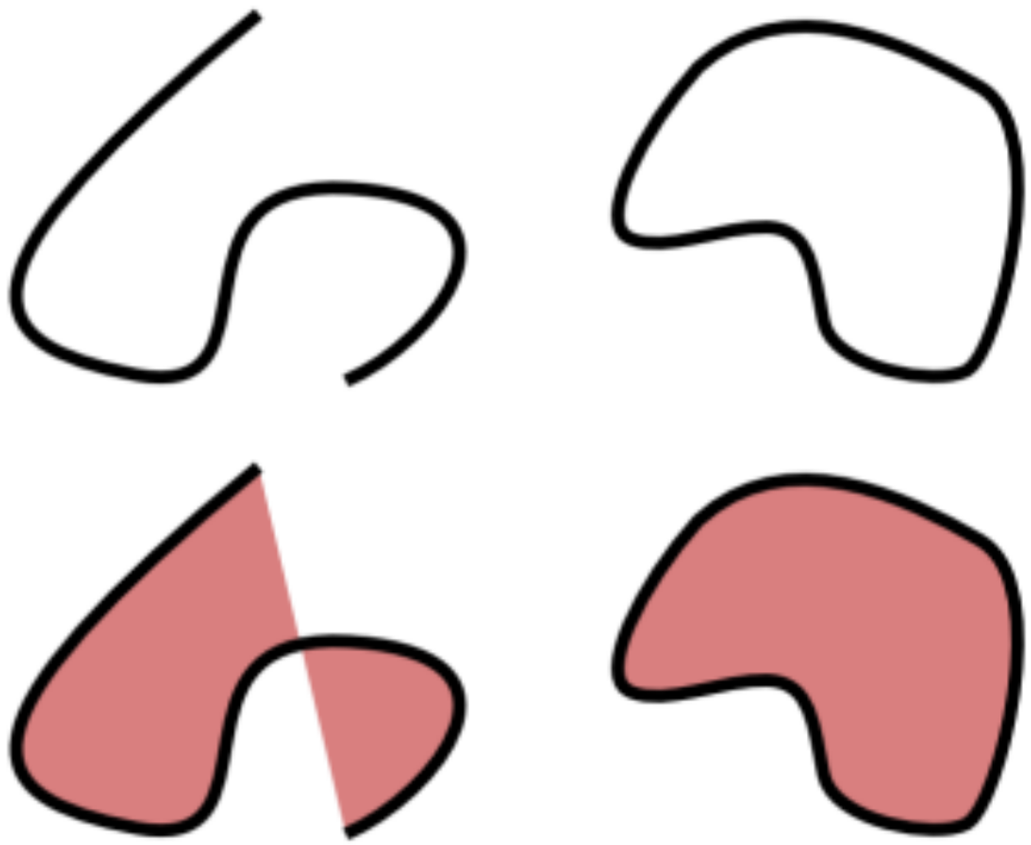


Vector Graphics Complex (VGC)
SIGGRAPH'14



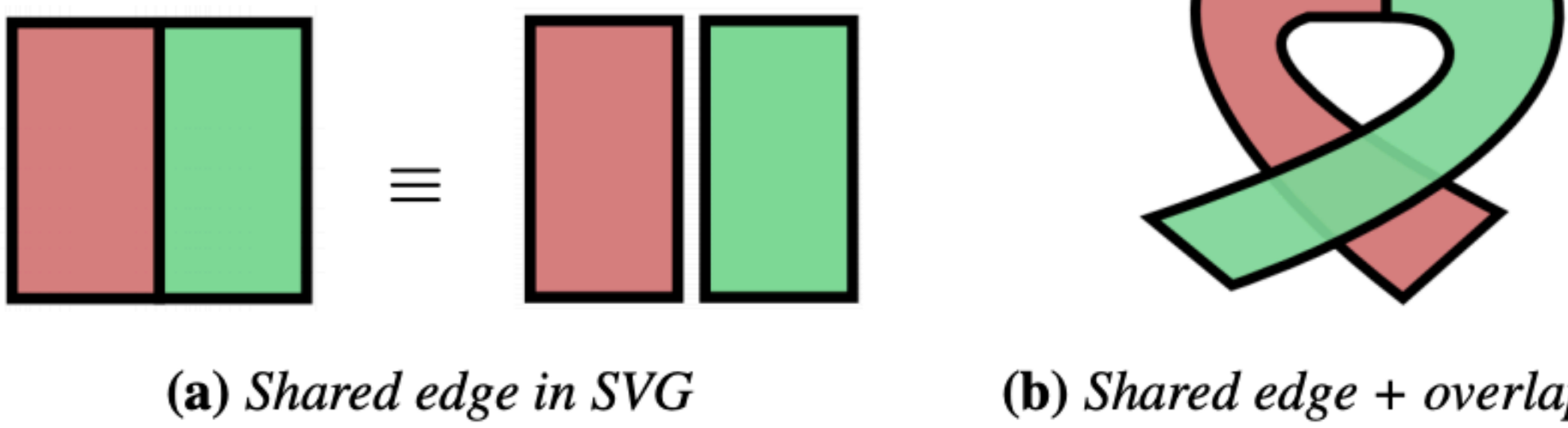
Vector Animation Complex (VAC)
SIGGRAPH'15

# Motivations



SVG:
A set of four basic primitives.

# Motivations



**(a)** *Shared edge in SVG* **(b)** *Shared edge + overlapping*

Redundant to create;
No partial layering;
Hard to edit;
Hard to animate.

# Motivations

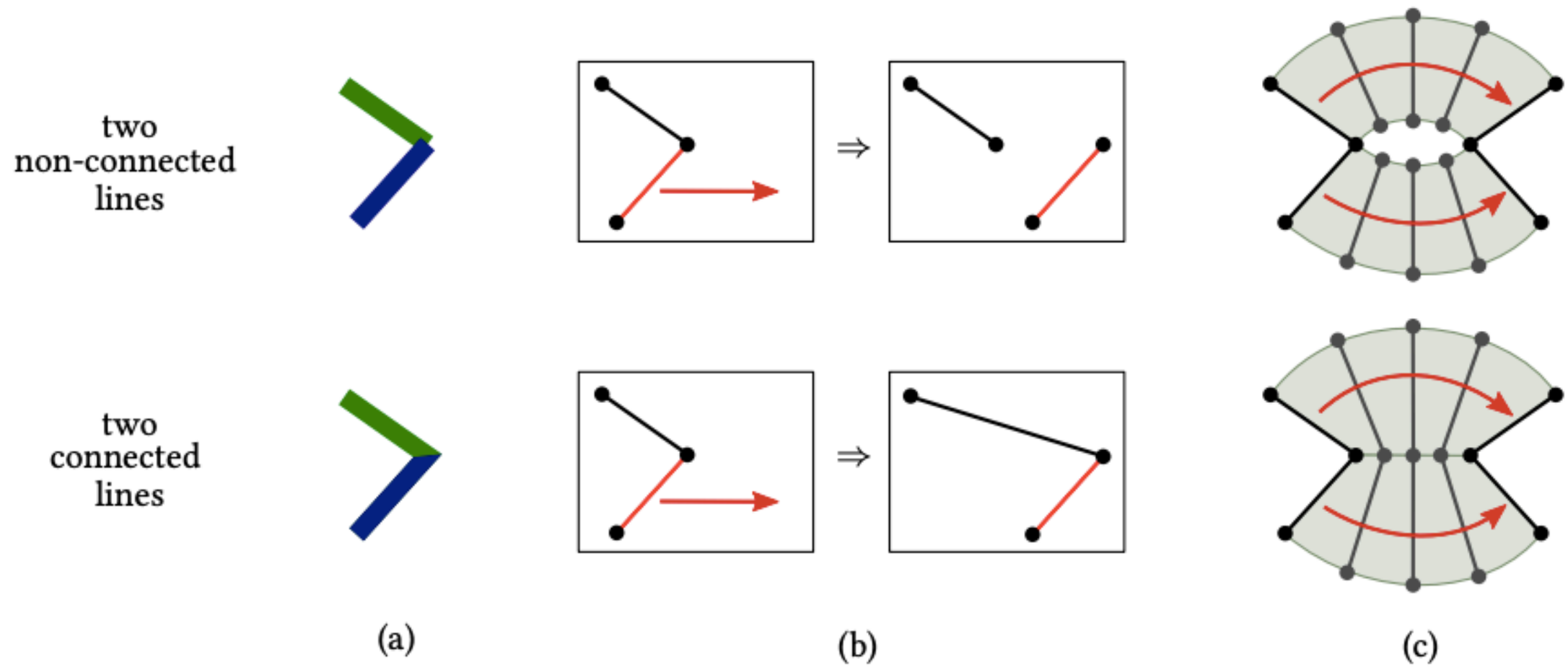
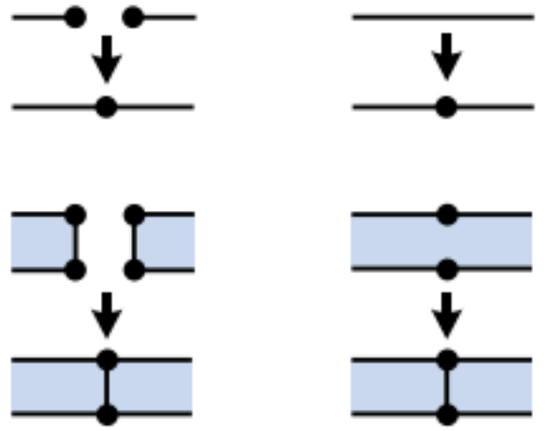



**Figure 3.2:** *Whether or not two lines are connected affects: (a) rendering, such as here the rendering of a Miter join; (b) user interaction, such as here a drag-and-drop action of one of the two lines; and (c) keyframe interpolation.*

# Motivations

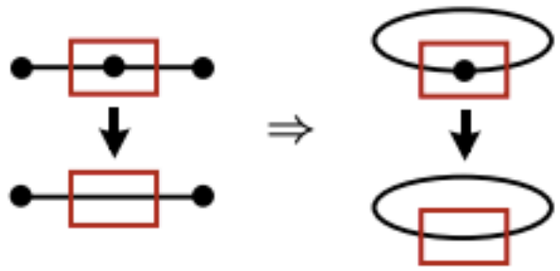
**A. Basic Primitives** At the very least, the following cells must be supported: vertices (single points in space); open edges (open curves starting and ending at a vertex); and triangles (surfaces homeomorphic to disks, bounded by three edges).

**B. Basic Topological Operators** Any two vertices can be glued. Any two edges can be glued using any of the two possible directions. Any edge can be cut by inserting an additional vertex. Any face can be cut by inserting an additional open edge starting and ending at existing boundary vertices.

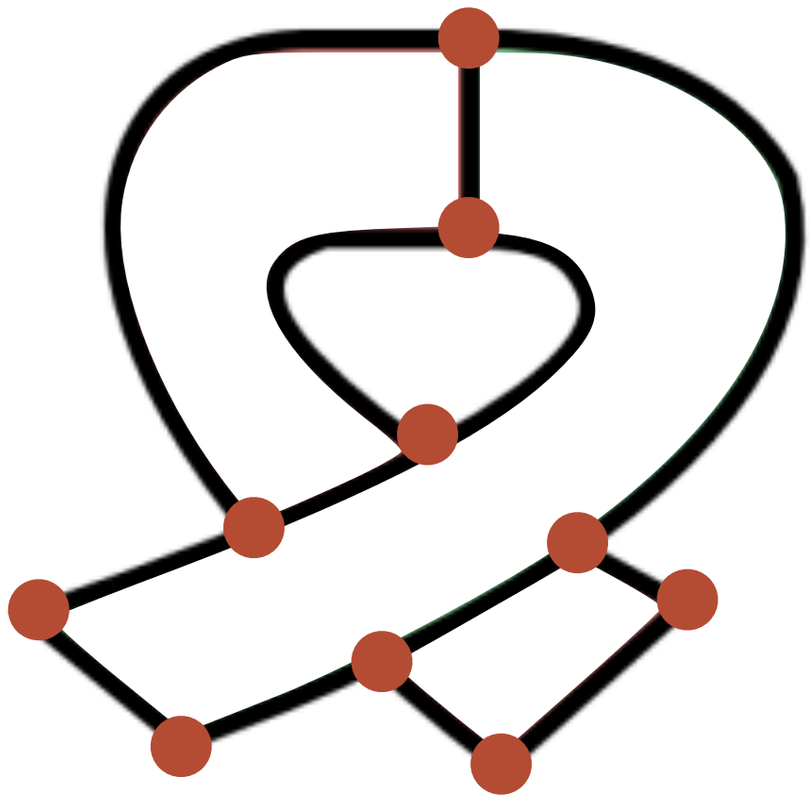


**C. Operator Invertibility** The inverse of any valid operator is also a valid operator.

**D. Operator Locality** The validity of any topological operator (i.e., whether or not it is allowed to apply it) only depends on local topological properties.



# Stroke Graph

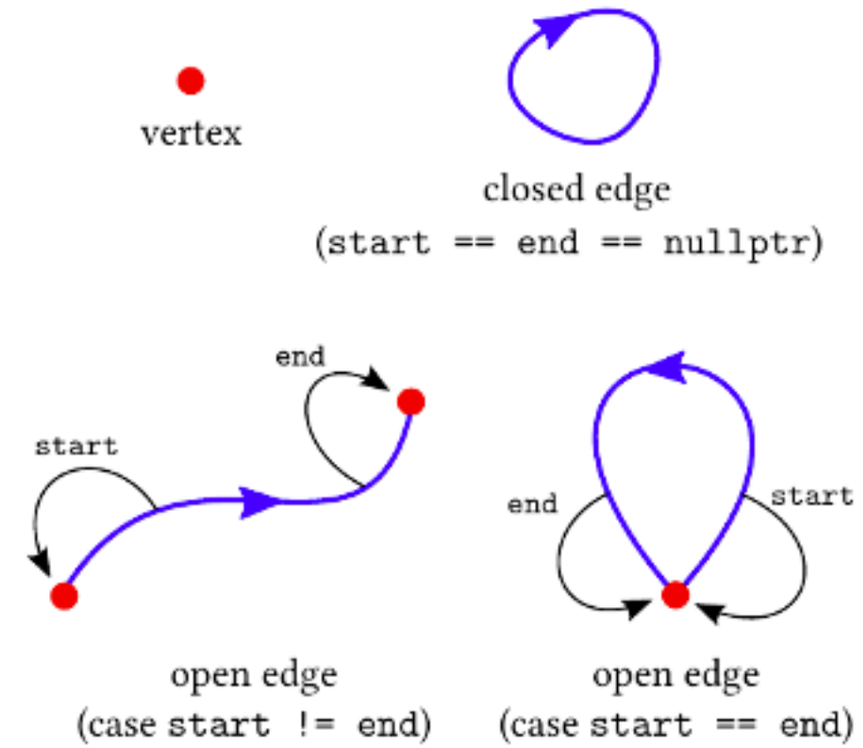


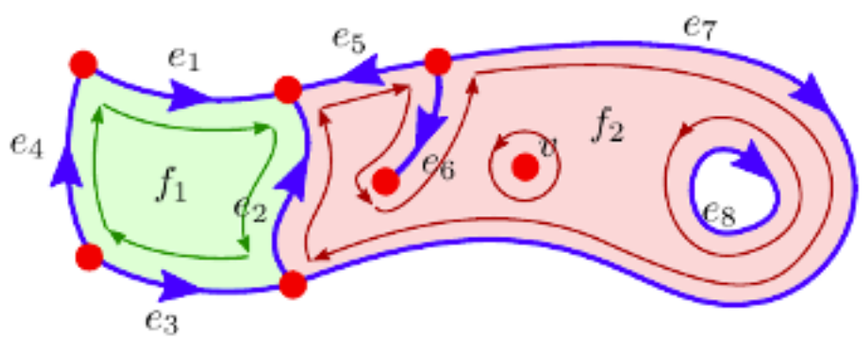
A graph (V, E), where each edge is a stroke.

[Whited et al. 2010]

# Solution: VGC

```
1  class Cell {
2      std::unordered_set<Cell*> star;
3  };
4
5  class Vertex: public Cell {
6      Point p;
7  };
8
9  class Edge: public Cell {
10     Vertex *start, *end;
11     DirectedCurve curve;
12 };
13
14 class Halfedge {
15     Edge *edge;
16     bool direction;
17 };
18
19 class Cycle {
20     Vertex *steiner;
21     std::vector<Halfedge> halfedges;
22 };
23
24 class Face: public Cell {
25     std::vector<Cycle> cycles;
26 };
27
28 class VGC {
29     std::unordered_set<Cell*> cells;
30 };
```



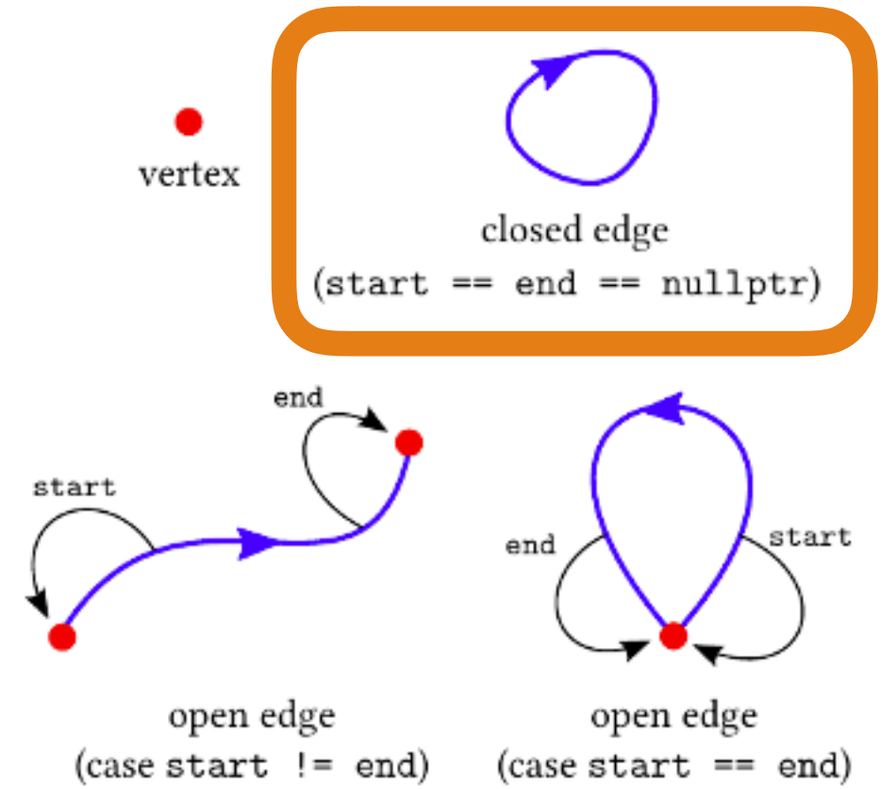

two incident faces

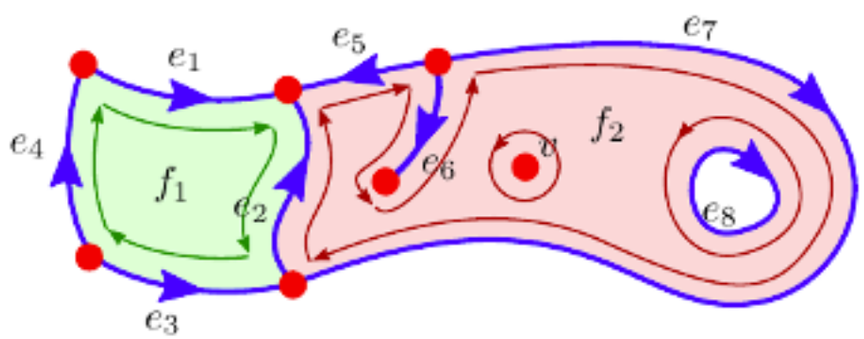
$f_1$->cycles = $[\, e_1 e_2^{-1} e_3^{-1} e_4 \,]$

$f_2$->cycles = $[\, e_2 e_5^{-1} e_6 e_6^{-1} e_7;\ v;\ e_8^{-1} \,]$

# Solution: VGC

```
1  class Cell {
2      std::unordered_set<Cell*> star;
3  };
4
5  class Vertex: public Cell {
6      Point p;
7  };
8
9  class Edge: public Cell {
10     Vertex *start, *end;
11     DirectedCurve curve;
12 };
13
14 class Halfedge {
15     Edge *edge;
16     bool direction;
17 };
18
19 class Cycle {
20     Vertex *steiner;
21     std::vector<Halfedge> halfedges;
22 };
23
24 class Face: public Cell {
25     std::vector<Cycle> cycles;
26 };
27
28 class VGC {
29     std::unordered_set<Cell*> cells;
30 };
```



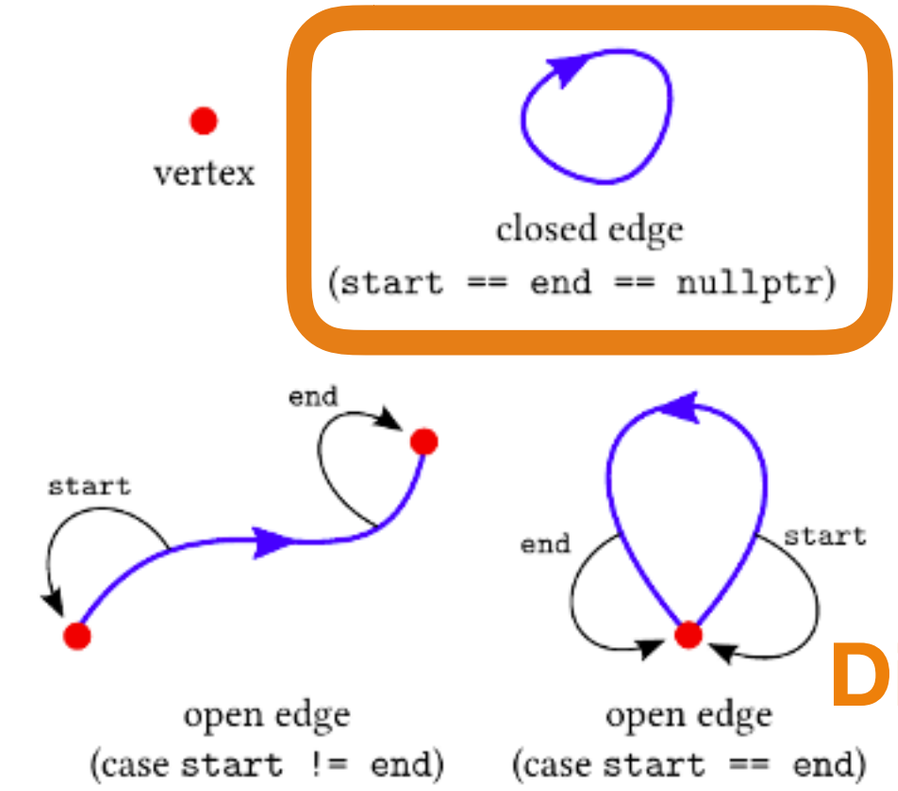

two incident faces

$f_1$->cycles = $[\, e_1 e_2^{-1} e_3^{-1} e_4 \,]$

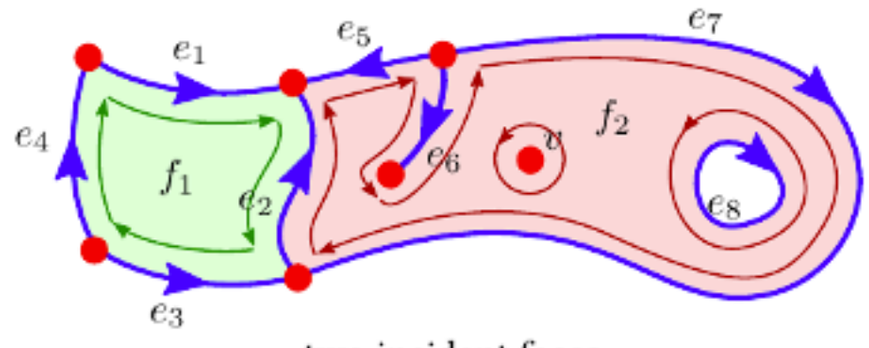
$f_2$->cycles = $[\, e_2 e_5^{-1} e_6 e_6^{-1} e_7;\ v;\ e_8^{-1} \,]$

# Solution: VGC

```
1  class Cell {
2      std::unordered_set<Cell*> star;
3  };
4
5  class Vertex: public Cell {
6      Point p;
7  };
8
9  class Edge: public Cell {
10     Vertex *start, *end;
11     DirectedCurve curve;
12 };
13
14 class Halfedge {
15     Edge *edge;
16     bool direction;
17 };
18
19 class Cycle {
20     Vertex *steiner;
21     std::vector<Halfedge> halfedges;
22 };
23
24 class Face: public Cell {
25     std::vector<Cycle> cycles;
26 };
27
28 class VGC {
29     std::unordered_set<Cell*> cells;
30 };
```




Different!



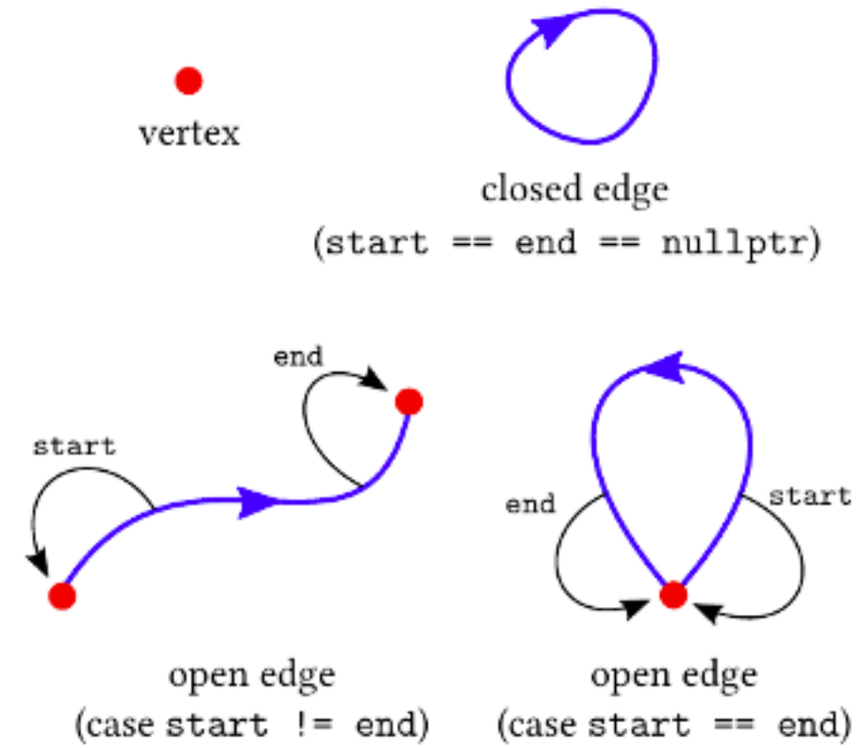

two incident faces

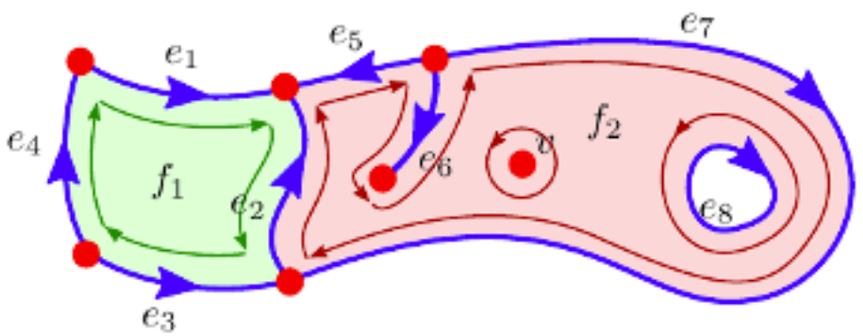
$f_1$->cycles = [ $e_1 e_2^{-1} e_3^{-1} e_4$ ]

$f_2$->cycles = [ $e_2 e_5^{-1} e_6 e_6^{-1} e_7$; $v$; $e_8^{-1}$ ]

# Solution: VGC

```
1  class Cell {
2      std::unordered_set<Cell*> star;
3  };
4
5  class Vertex: public Cell {
6      Point p;
7  };
8
9  class Edge: public Cell {
10     Vertex *start, *end;
11     DirectedCurve curve;
12 };
13
14 class Halfedge {
15     Edge *edge;
16     bool direction;
17 };
18
19 class Cycle {
20     Vertex *steiner;
21     std::vector<Halfedge> halfedges;
22 };
23
24 class Face: public Cell {
25     std::vector<Cycle> cycles;
26 };
27
28 class VGC {
29     std::unordered_set<Cell*> cells;
30 };
```



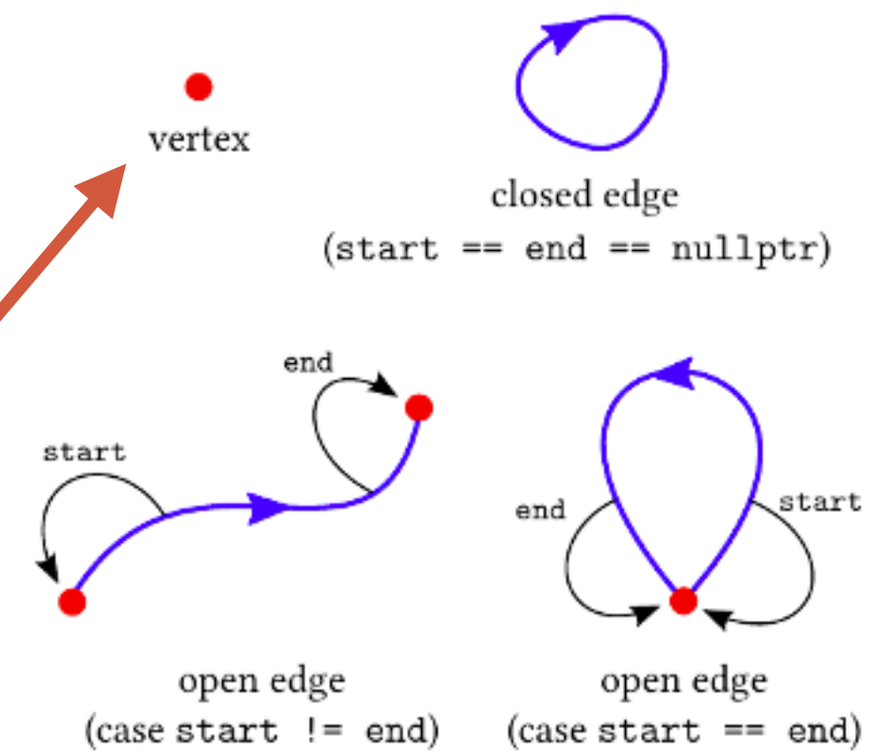



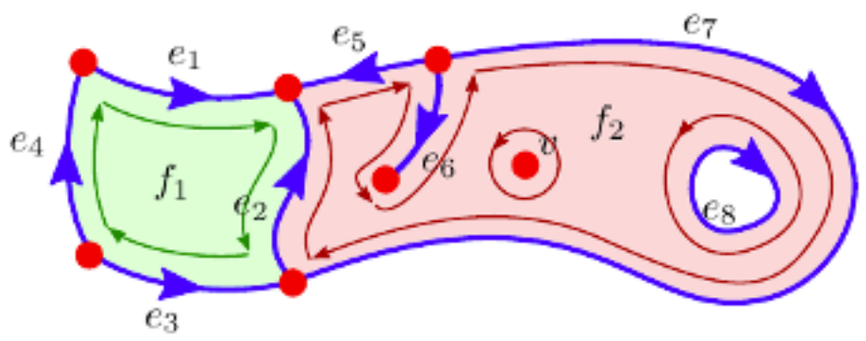

$f_1$->cycles = $[\, e_1 e_2^{-1} e_3^{-1} e_4 \,]$

$f_2$->cycles = $[\, e_2 e_5^{-1} e_6 e_6^{-1} e_7;\ v;\ e_8^{-1} \,]$

# Solution: VGC

```
1  class Cell {
2      std::unordered_set<Cell*> star;
3  };
4
5  class Vertex: public Cell {
6      Point p;
7  };
8
9  class Edge: public Cell {
10     Vertex *start, *end;
11     DirectedCurve curve;
12 };
13
14 class Halfedge {
15     Edge *edge;
16     bool direction;
17 };
18
19 class Cycle {
20     Vertex *steiner;
21     std::vector<Halfedge> halfedges;
22 };
23
24 class Face: public Cell {
25     std::vector<Cycle> cycles;
26 };
27
28 class VGC {
29     std::unordered_set<Cell*> cells;
30 };
```



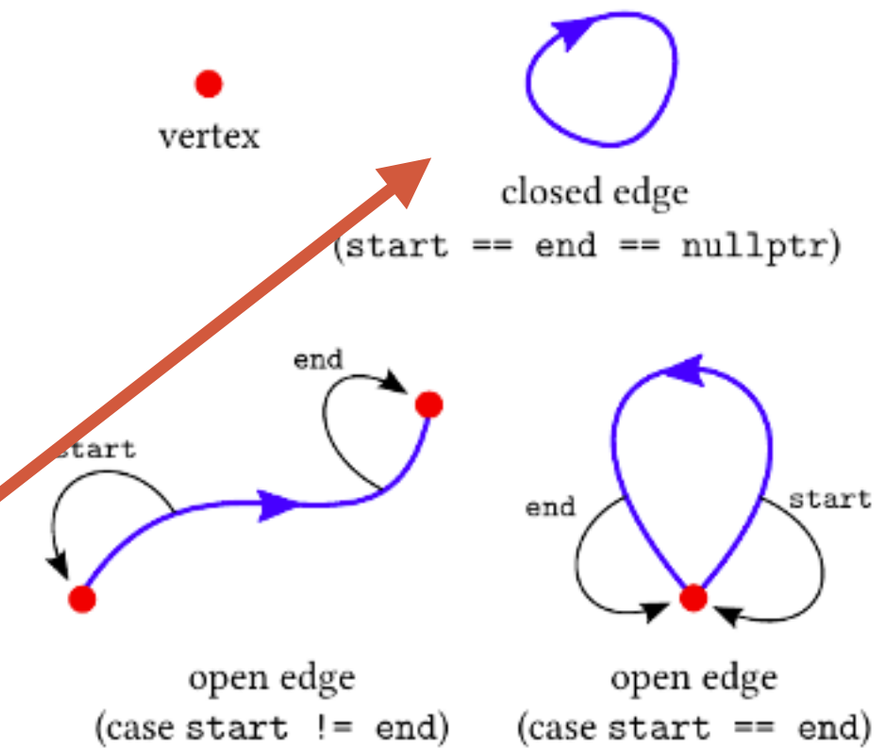



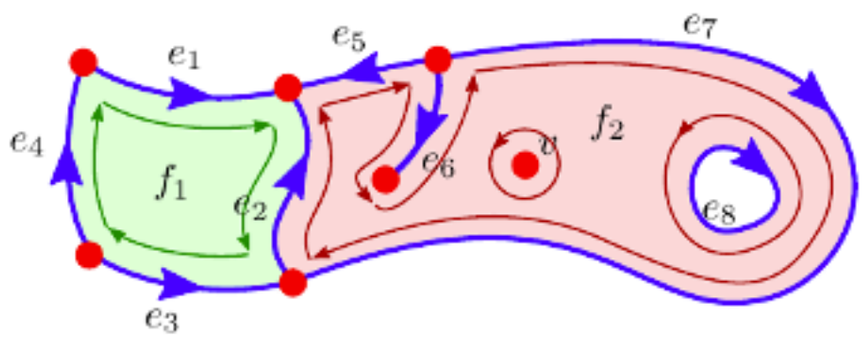

$f_1$->cycles = $[\, e_1 e_2^{-1} e_3^{-1} e_4 \,]$

$f_2$->cycles = $[\, e_2 e_5^{-1} e_6 e_6^{-1} e_7;\ v;\ e_8^{-1} \,]$

# Solution: VGC

```
1  class Cell {
2      std::unordered_set<Cell*> star;
3  };
4
5  class Vertex: public Cell {
6      Point p;
7  };
8
9  class Edge: public Cell {
10     Vertex *start, *end;
11     DirectedCurve curve;
12 };
13
14 class Halfedge {
15     Edge *edge;
16     bool direction;
17 };
18
19 class Cycle {
20     Vertex *steiner;
21     std::vector<Halfedge> halfedges;
22 };
23
24 class Face: public Cell {
25     std::vector<Cycle> cycles;
26 };
27
28 class VGC {
29     std::unordered_set<Cell*> cells;
30 };
```




$f_1$->cycles = $[\, e_1 e_2^{-1} e_3^{-1} e_4 \,]$

$f_2$->cycles = $[\, e_2 e_5^{-1} e_6 e_6^{-1} e_7;\ v;\ e_8^{-1} \,]$

# Supported Operations

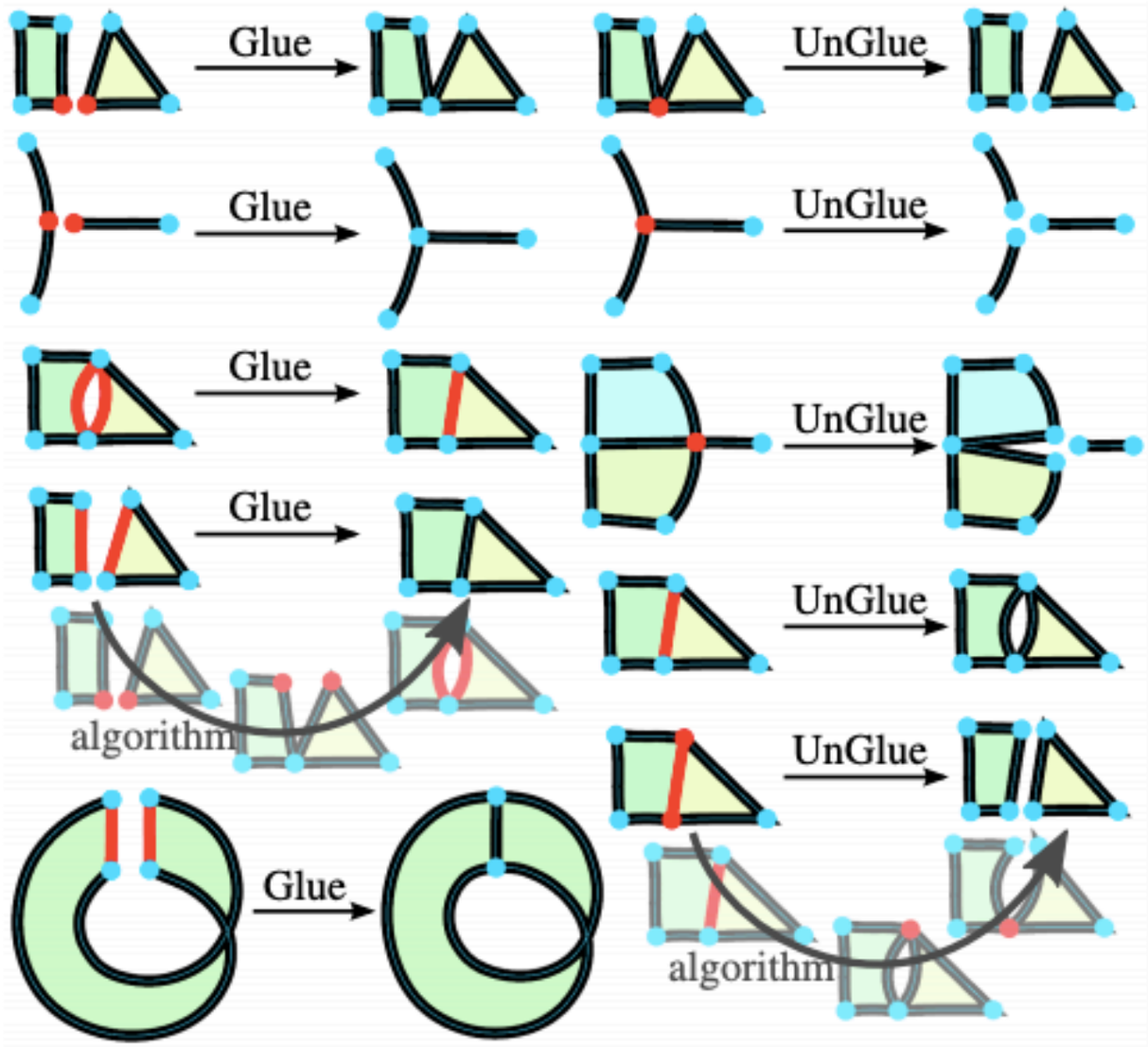



**Figure 14:** *Examples of glue and unglue operations on vertices, edges, and a set of cells (bottom-right).*

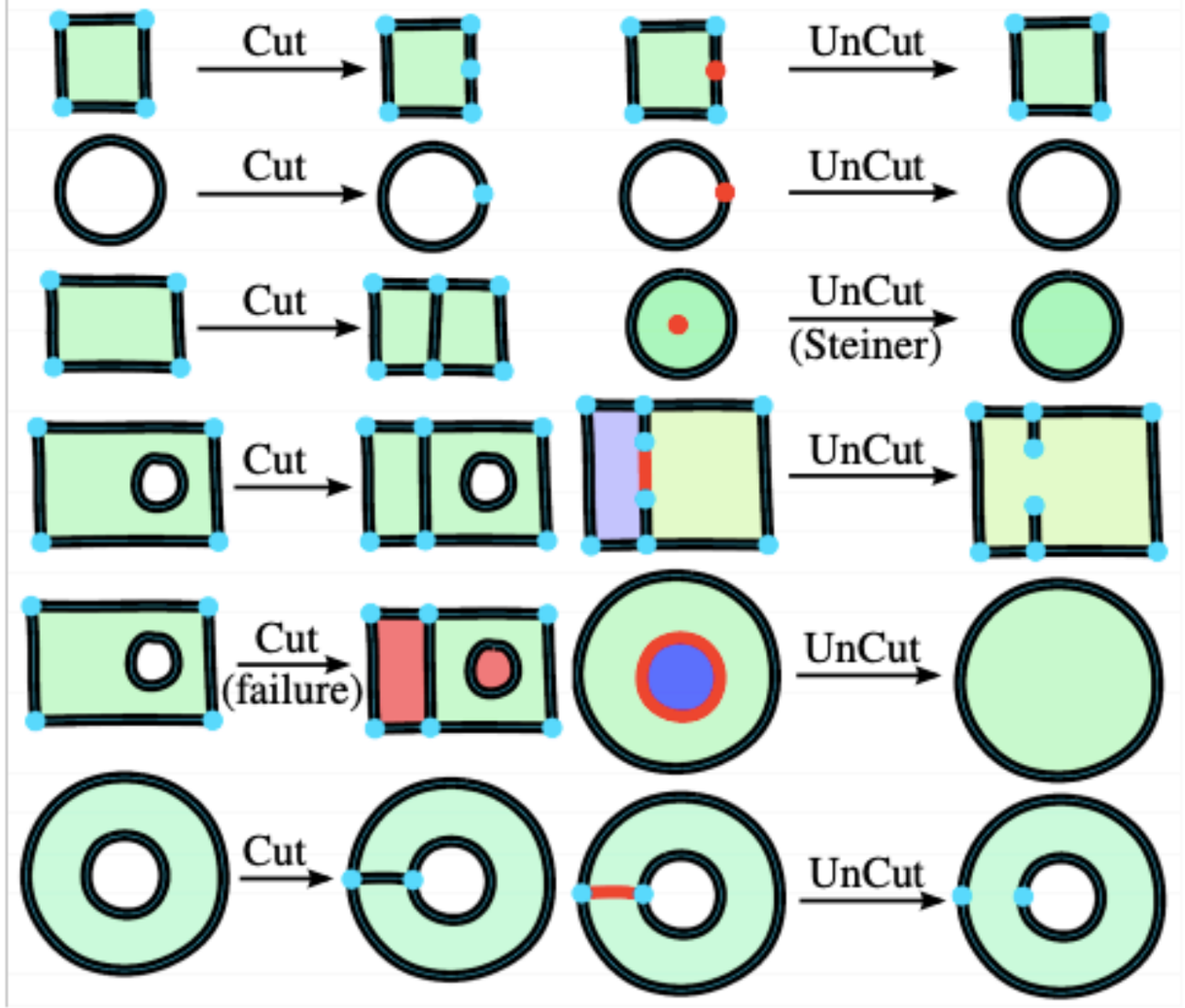
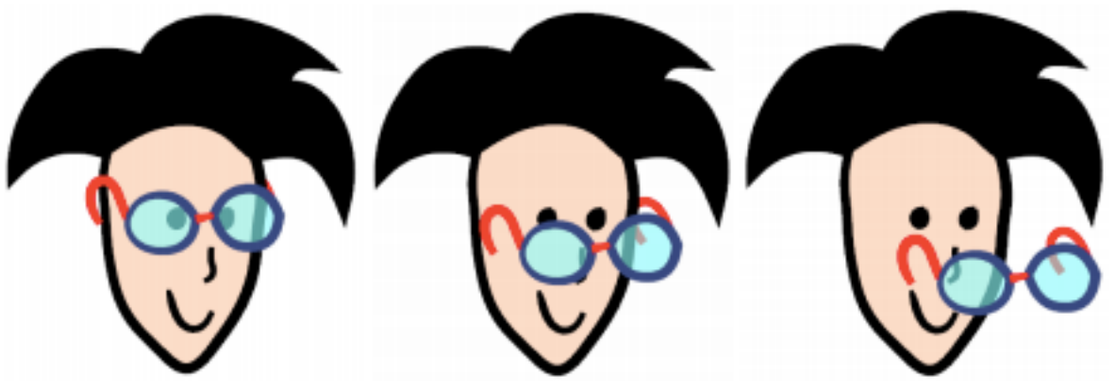



**Figure 15:** *Examples of cut and uncut operations on vertices and edges. The third operation on the right column illustrates uncutting a Steiner vertex from a face. It is topology equivalent to the fifth operation on the same column. The fifth example on left column is a failure case, when the cut algorithm transfers the "hole cycle" to the wrong face (shown in red). This happens because the cut operator is actually ambiguous and disambiguation require geometric heuristics to capture the user's intent.*

# Results



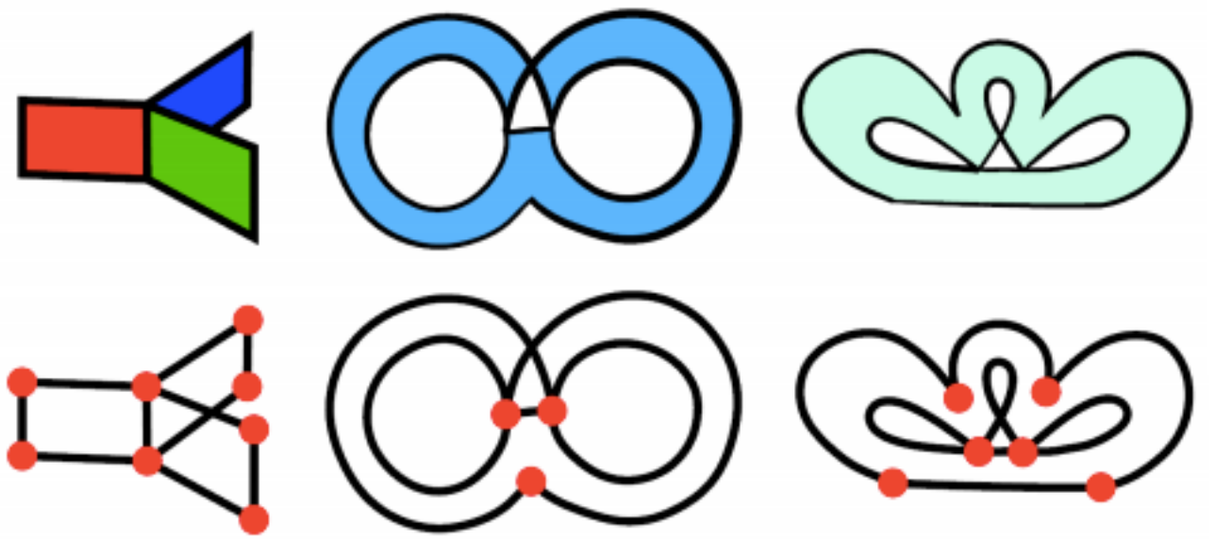


**Figure 19:** *Examples of non-manifold topologies where an edge has three "face uses". These uses may be by the same face (middle and right), and part of a hole (right)*

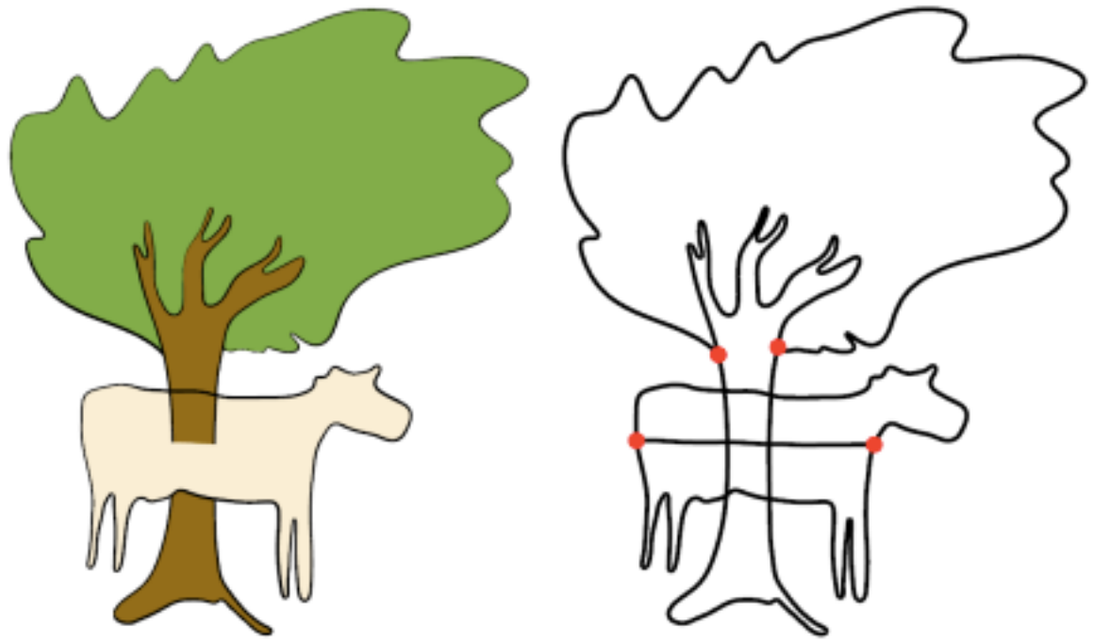
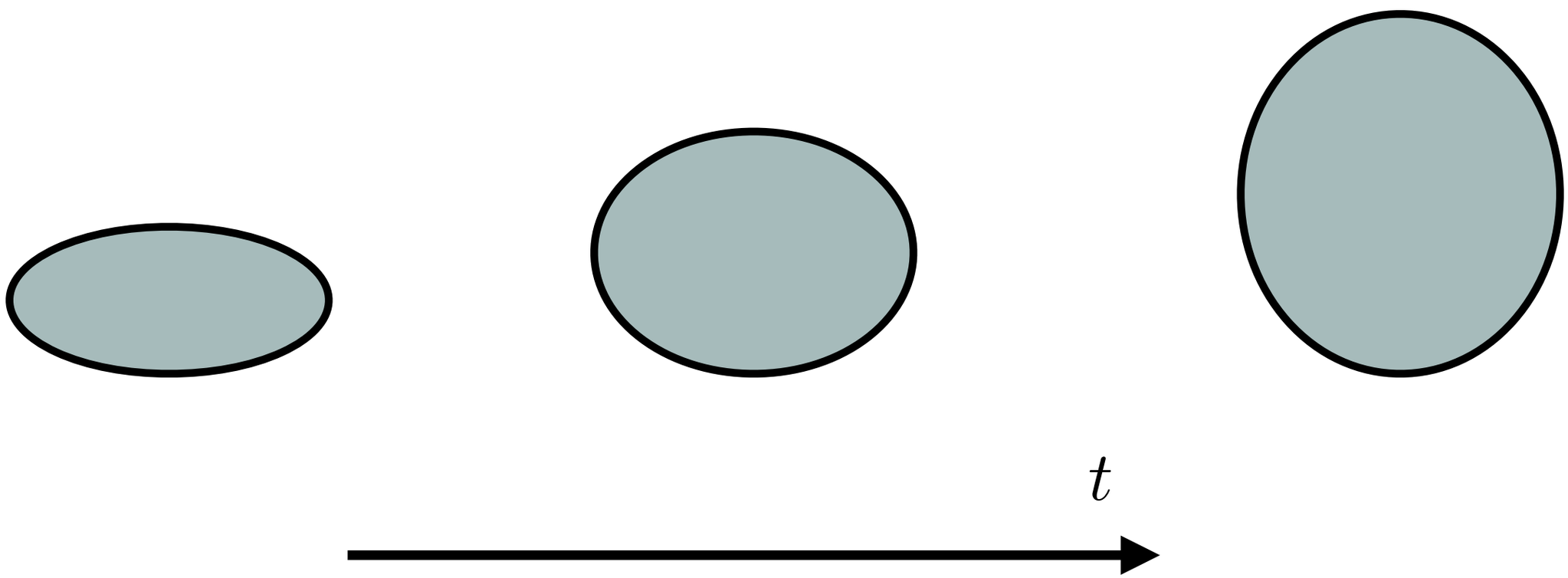


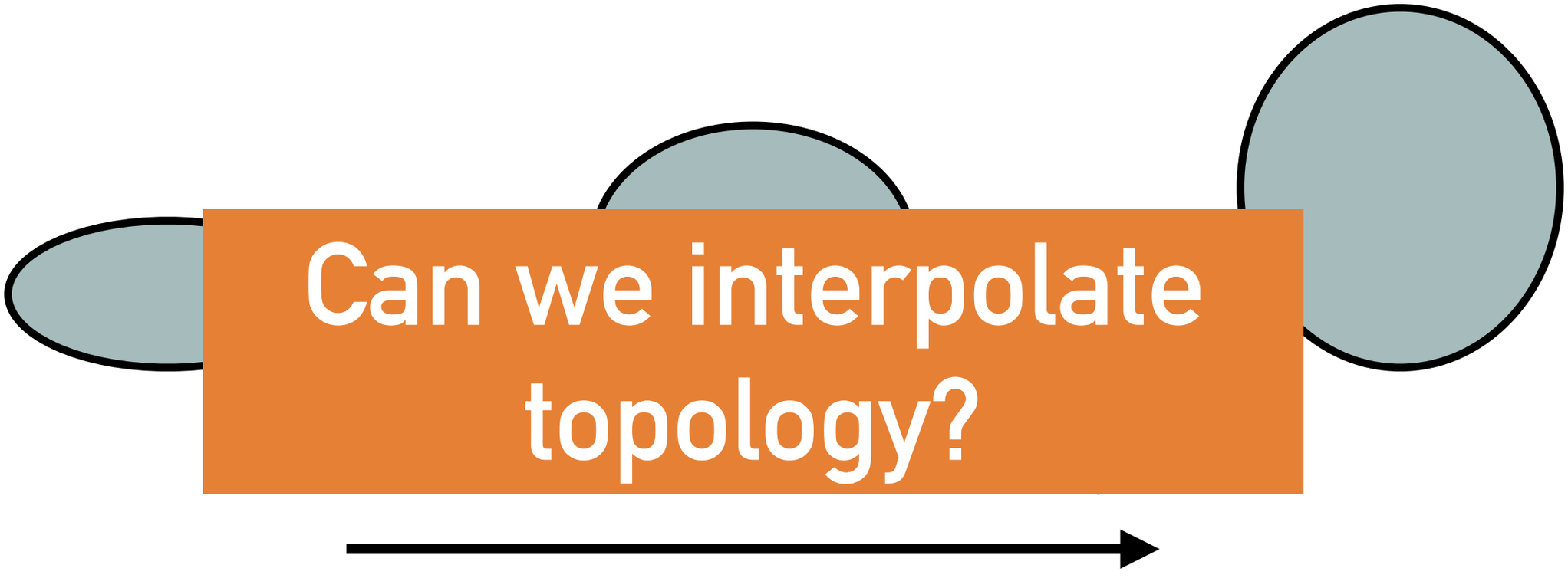
**Figure 20:** *A user experimenting with the possibilities offered by invisibe cuts and depth ordering.*

# 2D Animation



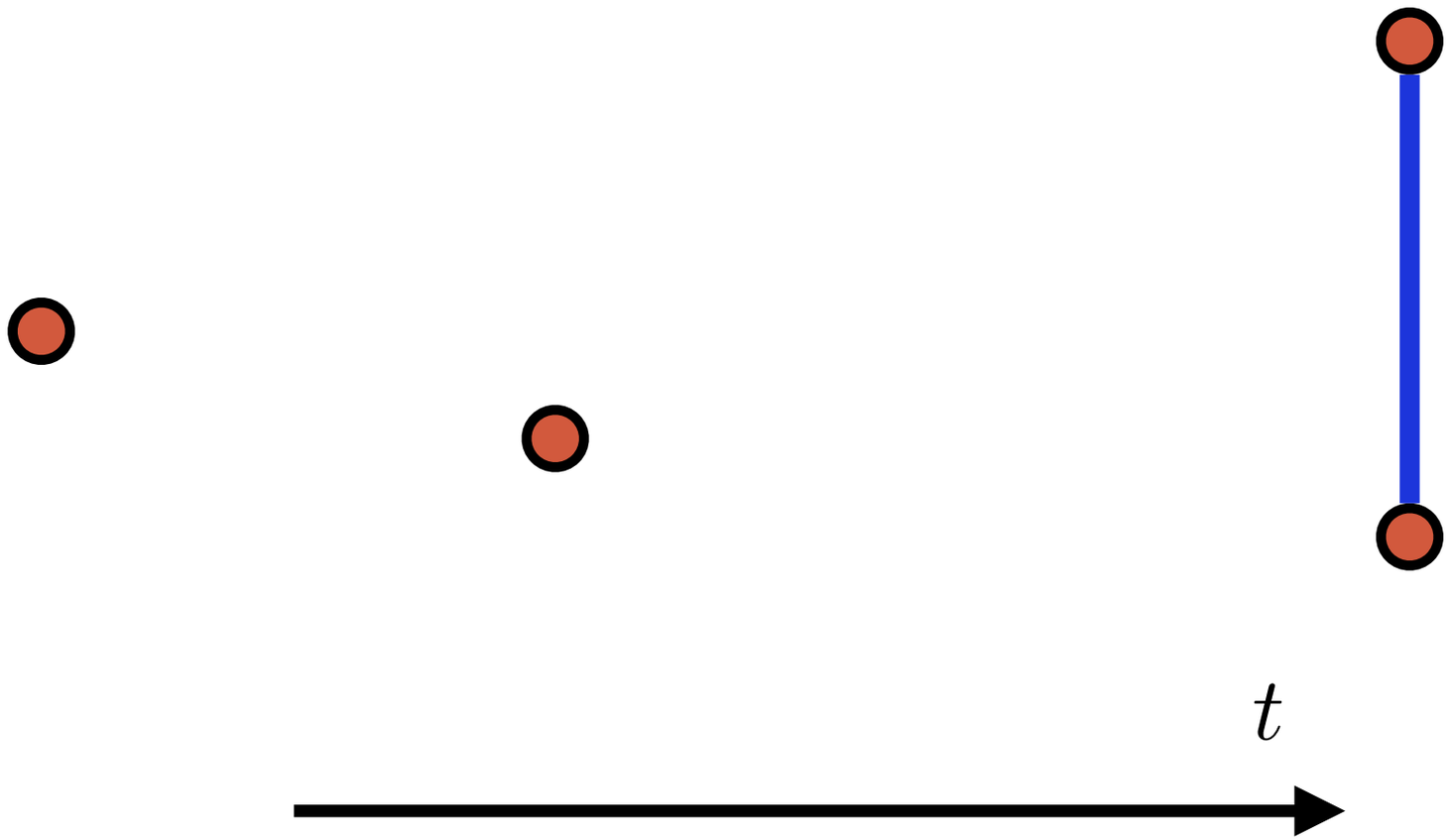

**Standard 2D animation:**
**Geometric Interpolation**

# 2D Animation




**Standard 2D animation:**
**Geometric Interpolation**

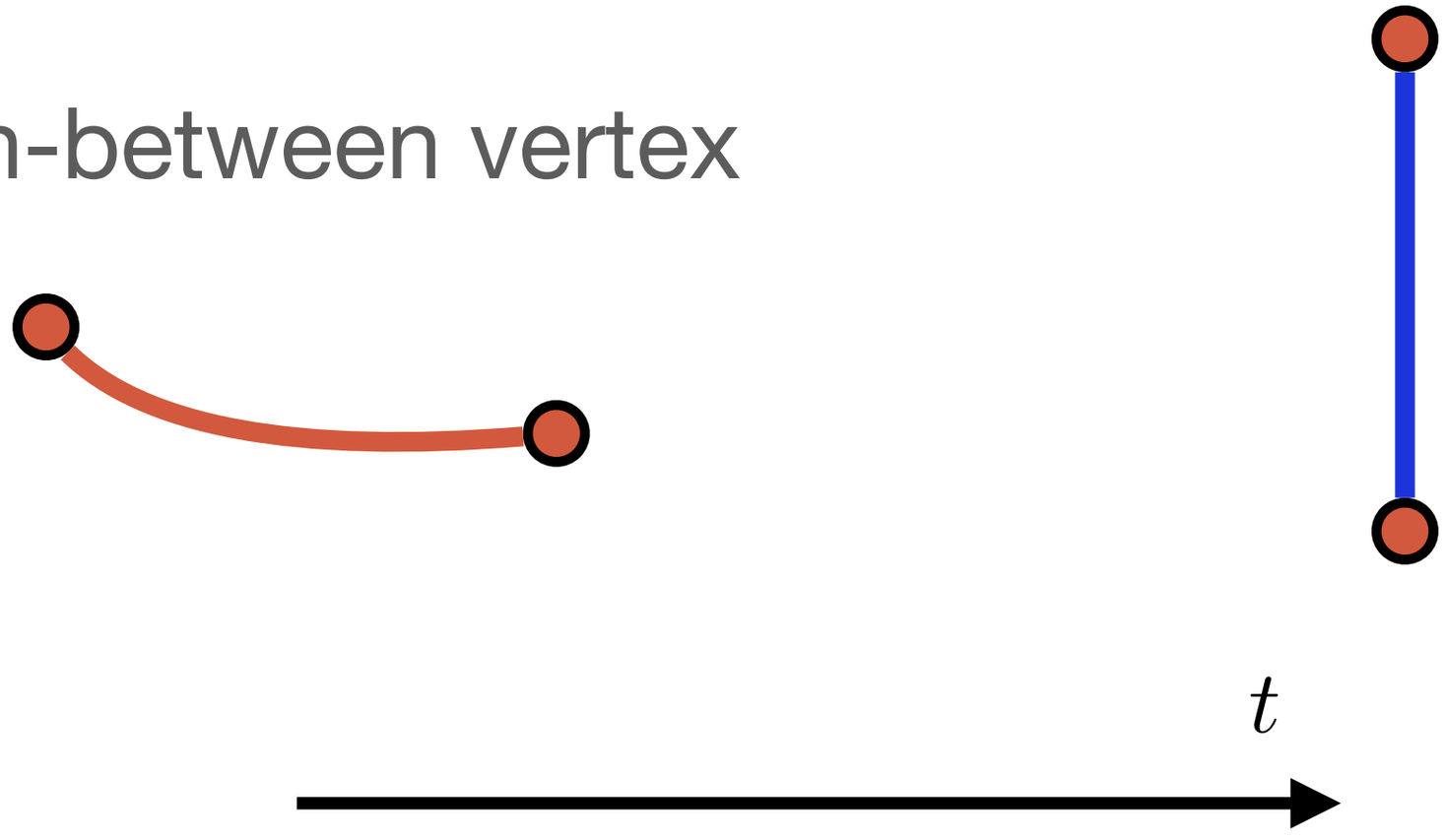
# Solution: VAC




Keyframes: VGC

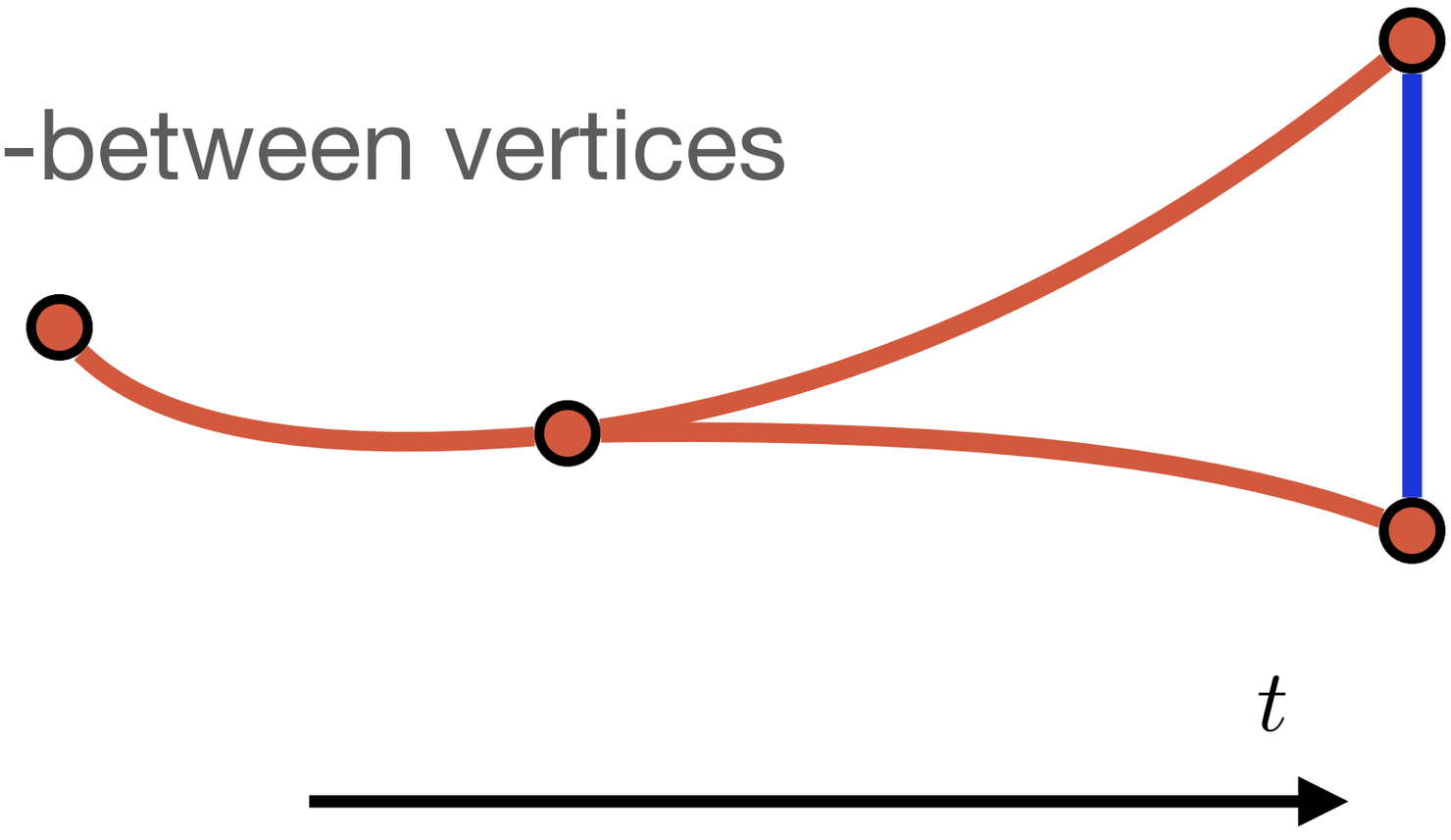
# Solution: VAC

In-between vertex

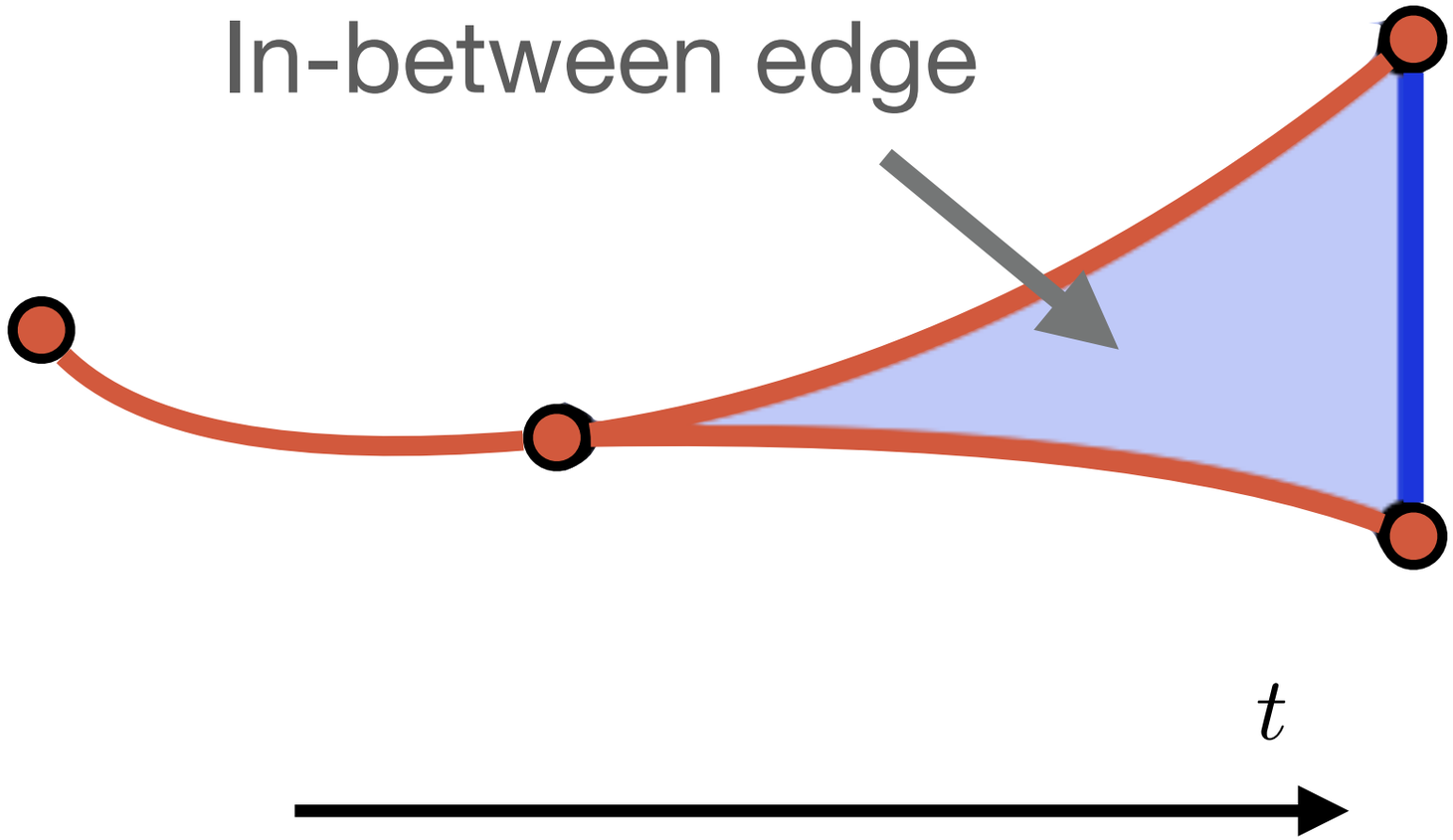


# Solution: VAC

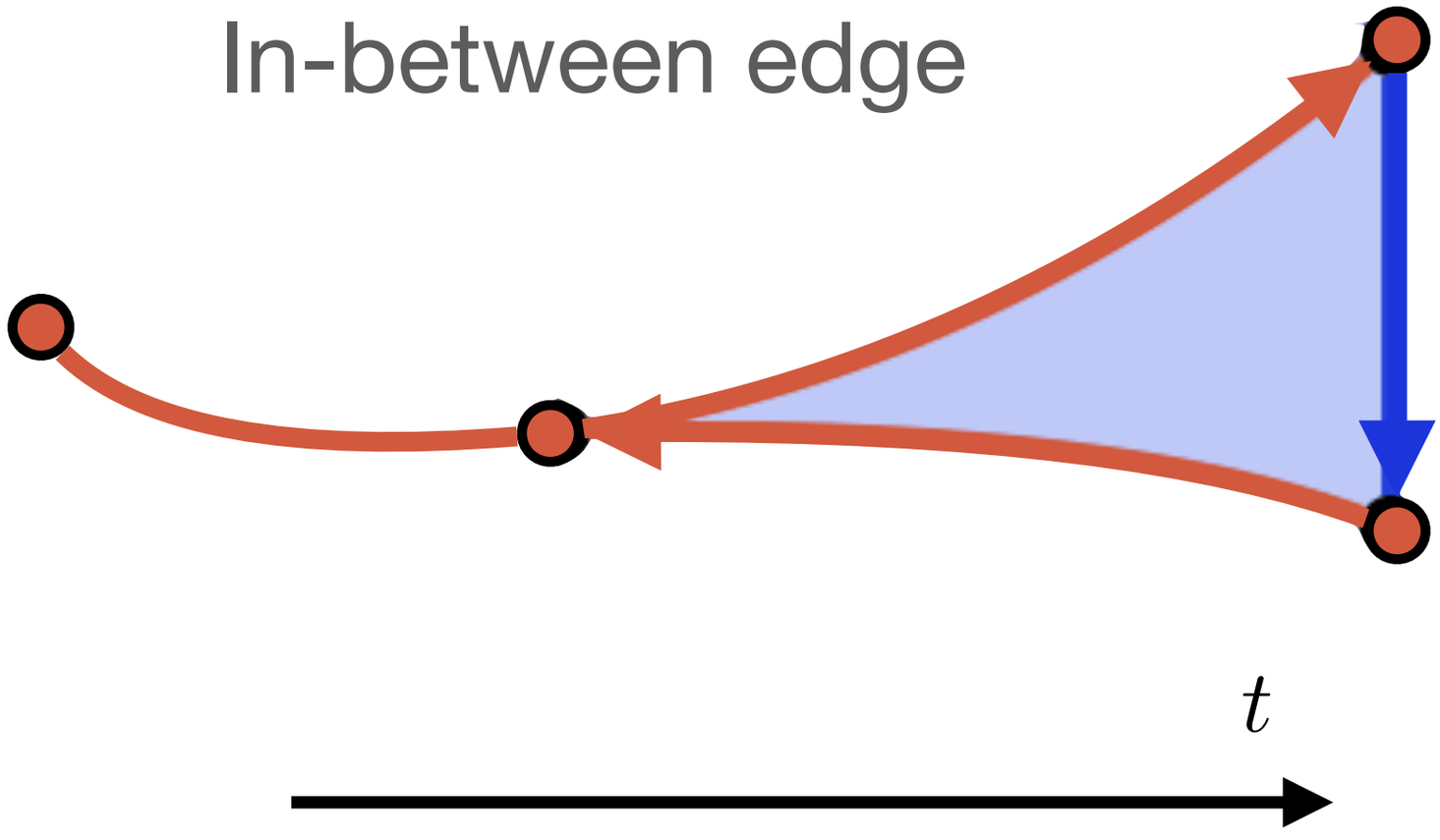
In-between vertices



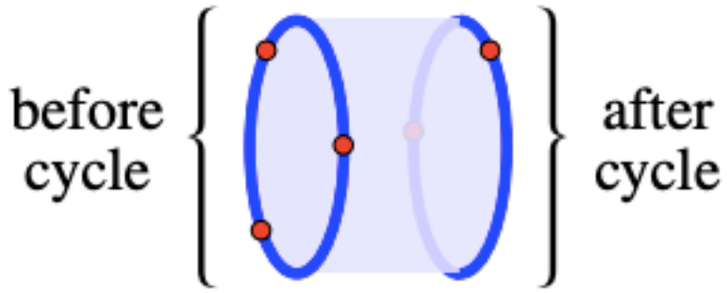
# Solution: VAC



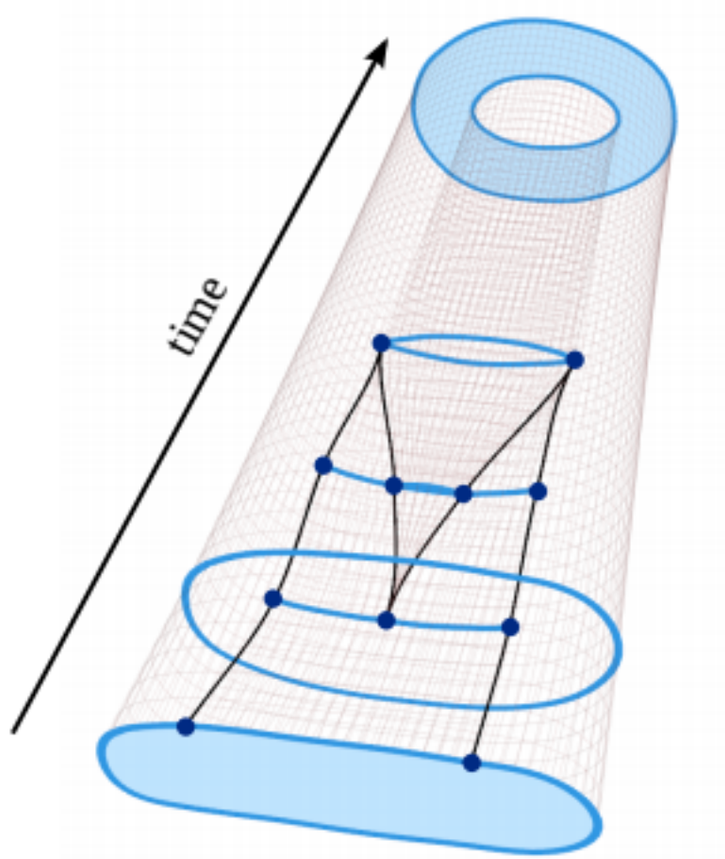
# Solution: VAC



# Solution: VAC



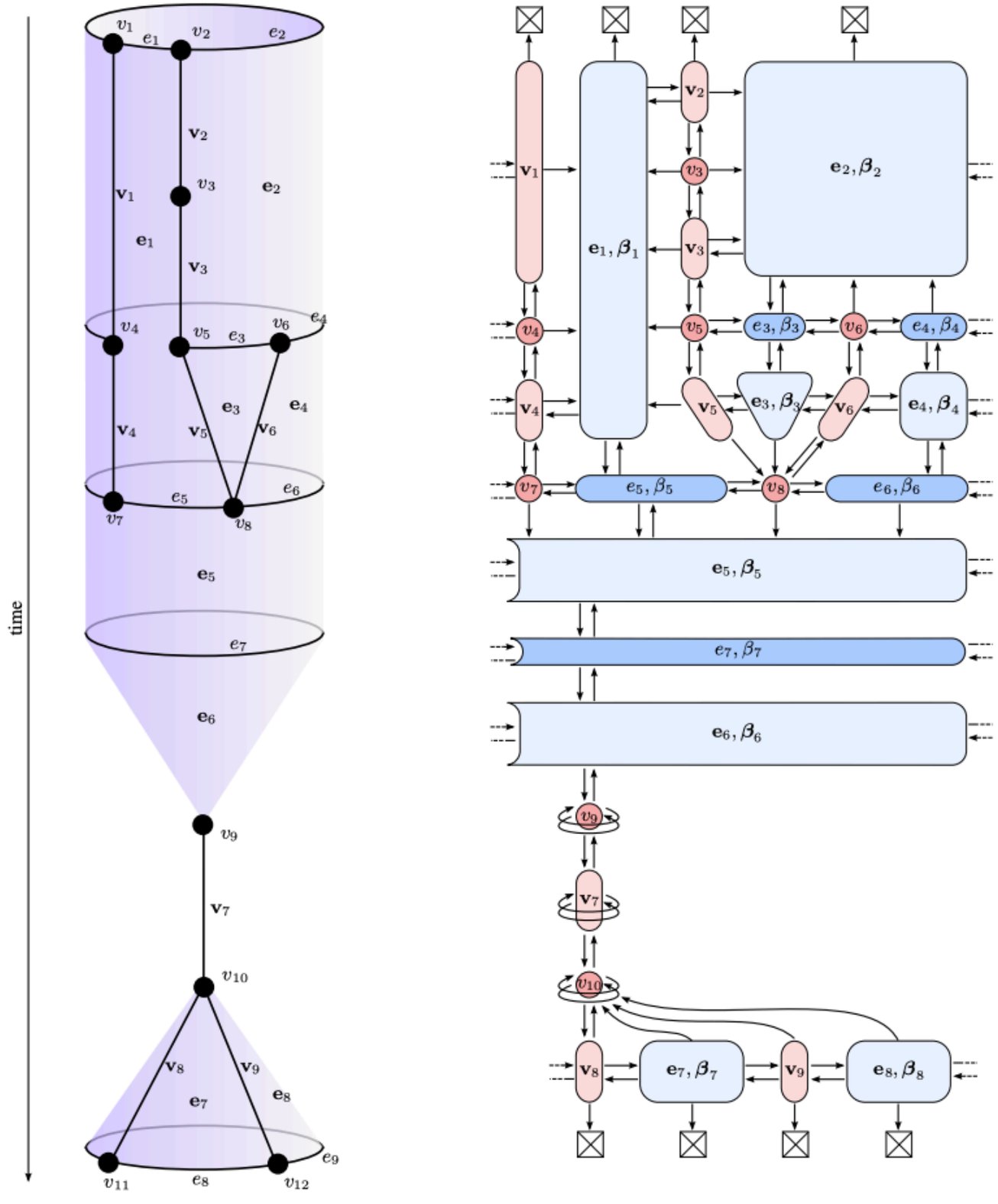

**Inbetween closed edge**




**Space-time visualization**

Interpolate close edges and faces.

# 2D Double Linked-List



# Results

https://www.borisdalstein.com/research/vac/

# Comments

- The real insight of Boris's work is in his technical report and thesis. It's a topological space called PCS complex, which is constructed by applying multiple (un)glue and (un)cut to basic primitives. And an exhaustive classification of the 19 different ways a face can be cut along an edge.
- Yet, the catch is PCS can't be rendered using the winding rule, thus VGC has to be defined by dropping the orientability and genus. The result is inevitable ambiguities in cutting (solved by unclear heuristics in practice).

# Comments (continued)

- Limitations:
  - No continuity constraint across joint;
  - No boolean operation;
  - No self-intersecting face (due to the same PCS rendering problem).
- I think there may be a connection between the VGC and the NPR rendering of a 3D model. But how VGC may benefit the line drawing rendering is still unclear…

# THANKS!