

# Arrangements on Parametric Surfaces I: General Framework and Infrastructure

Eric Berberich, Efi Fogel, Dan Halperin,  
Kurt Mehlhorn and Ron Wein

**Abstract.** We introduce a framework for the construction, maintenance, and manipulation of arrangements of curves embedded on certain two-dimensional orientable parametric surfaces in three-dimensional space. The framework applies to planes, cylinders, spheres, tori, and surfaces homeomorphic to them. We reduce the effort needed to generalize existing algorithms, such as the sweep line and zone traversal algorithms, originally designed for arrangements of bounded curves in the plane, by extensive reuse of code. We have realized our approach as the CGAL package `Arrangement_on_surface_2`. We define a compact and modular interface for our framework; for a given application a required small subset of the interface can be identified. Then, only this subset must be implemented. A companion paper describes concretizations for several types of surfaces and curves embedded on them, and applications. This is the first implementation of a generic algorithm that can handle arrangements on a large class of parametric surfaces.

**Mathematics Subject Classification (2010).** Primary 68U05; Secondary 14Q10.

**Keywords.** computational geometry, arrangement of curves, parametric surface, CGAL, robust geometric computing.

## 1. Introduction

We are given a surface  $S$  in  $\mathbb{R}^3$  and a set  $\mathcal{C}$  of curves embedded on this surface. The curves subdivide  $S$  into cells of dimension 0 (*vertices*), 1 (*edges*), and 2 (*faces*). This subdivision is the *arrangement*  $\mathcal{A}(\mathcal{C})$  induced by  $\mathcal{C}$  on  $S$ . We present a generic framework for the construction, maintenance, and manipulation of arrangements

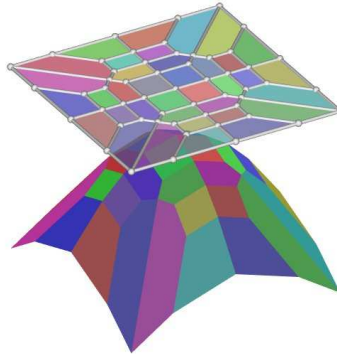
---

This work has been supported in part by the Israel Science Foundation (grant no. 236/06), by the German-Israeli Foundation (grant no. 969/07), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University. A preliminary version of this paper [8] has appeared in the Proceedings of the 15th Annual European Symposium on Algorithms (ESA 2007).

(with a finite number of cells) embedded on two-dimensional orientable parametric surfaces such as planes, cylinders, spheres, tori, and surfaces homeomorphic to them. Such arrangements have many theoretical and practical applications [2, 7, 18, 22]. Our work is conceptual — the definition of the framework — and practical — the implementation of the framework. The latter is provided as the CGAL package `Arrangement_on_surface_2` [33] as of version 3.4, albeit only partially documented.<sup>1</sup> CGAL, the Computational Geometry Algorithms Library,<sup>2</sup> consists of generic and robust, yet efficient, implementation of geometric data-structures and algorithms [17]. Our package supports the construction of arrangements, insertion and removal of curves, iteration over all records (i. e., vertices, edges, and faces) and the records incident to a particular record. It also provides additional operations, such as point location and overlay computation. Other packages of CGAL based on the `Arrangement_on_surface_2` package provide additional functionality, such as Boolean set operations, envelope computation, and Voronoi diagram construction. Our framework has a compact and modular interface. For a given application a required small subset of the interface can be identified. Then, only this subset must be implemented. In this way, the package can be conveniently adapted to different scenarios. The algorithmic machinery is generic and provided by the package. We and others have already instantiated the package for different surfaces and curves; see Section 5 and the companion paper [7]. Examples are arrangement of lines in the plane, of arcs of great circles on the sphere [19, 20], of intersection curves between quadric surfaces and a fixed quadric [8], and of intersection curves between arbitrary algebraic surfaces and a fixed Dupin cyclide [10]. The torus is a Dupin cyclide.

The starting point for our work was the CGAL package for constructing and maintaining arrangements of bounded curves in the plane [32]. It could not handle unbounded curves directly. Rather, unbounded curves had to be clipped by the user in a preprocessing phase, so that no essential information about the arrangements (e. g., a finite intersection point) was lost. This solution is inconvenient. For example, the *minimization diagram* of a set of surfaces in  $\mathbb{R}^3$  results in a planar arrangement, where each face is labeled with the lowest surfaces above it [29].

Such an arrangement has, in general, several unbounded faces. However, an arrangement of bounded curves has only a single unbounded face. Naturally, preprocessing (enclosure in a bounding rectangle) and postprocessing allows one to



<sup>1</sup>The manual of the `Arrangement_on_surface_2` package in CGAL version 3.4 does not include material for non-planar surfaces. We expect that a near future release will include the complete documentation.

<sup>2</sup><http://www.cgal.org>

recover the unbounded faces. However, pre- and post-processing outside the package has the consequence that many nice functions of the package, for instance, point location or overlays, are no longer available. The figure on the previous page, for example, depicts the lower envelopes of 36 planes, which had to be converted into corresponding 36 patches *a priori*, before their lower envelope could be represented as an arrangement of bounded curves.

Our initial goal was to extend the package so that unbounded curves in the plane could be handled within the package, and hence the full functionality of the package would also be available for arrangements of such curves. Our solution carries further than that; it can also deal with arrangements on certain surfaces, such as spheres, cylinders, tori, and surfaces homeomorphic to them.

Following CGAL, our package adheres to the *generic-programming* paradigm, making extensive use of C++ class- and function-templates [4]. The paradigm uses a formal hierarchy of abstract requirements on data types called *concepts*. When a concept extends the requirements of another concept, the former is said to be a *refinement* of the latter. When a type meets the set of concept requirements, the type is a *model* of the concept. For readers unfamiliar with generic programming the following analogue from mathematics should be useful: a group is a concept and a specific group, for instance,  $(\mathbb{Z}, +)$ , is a model of this concept. Concepts correspond to template parameters, and models correspond to types (typically classes) that substitute template parameters of template classes or functions when instantiated. We also make use of *design patterns* [21]. A design pattern is a general solution to a commonly occurring problem in software design. The `Arrangement-on-surface-2` package, for example, applies the *observer* pattern to automatically notify a list of dependent components about structural changes of the arrangement data-structure; see [32] for more details.

**Related work:** Effective algorithms for manipulating arrangements of curves have been a topic of considerable interest in recent years, with an emphasis on exactness and efficiency of implementation [18]. Mehlhorn and Seel [28] propose a general framework for extending the sweep-line algorithm to unbounded curves; however, their implementation can only handle lines in the plane. Andrade and Stolfi [3] develop exact algorithms for manipulating circular arcs on a sphere. Halperin and Shelton [23] incrementally construct arrangements of circles on a sphere. Berberich *et al.* [9] construct arrangements of intersection curves of quadric surfaces with a fixed reference quadric. They maintain two arrangements, one for the lower part of the reference quadric and one for its upper part. This strategy requires a postprocessing step, which is not implemented. Our approach avoids the need for a postprocessing step. Cazals and Lorient [12, 14] have developed a software package that computes exact arrangements of circles on a sphere. Their software is specialized for the spherical case. Hijazi and Breuel [25] compute arrangements induced by implicit curves using a subdivision method and interval arithmetic, and Milenkovic and Sacks [30] compute arrangements using approximations. In contrast, Eigenwillig and Kerber [16] describe an exact and complete approach to

compute arrangements of algebraic curves. Their most recent implementation uses our framework.

**Outline:** This paper is structured as follows. In Section 2 we review the arrangement framework for bounded curves in the plane. In Section 3 we generalize the framework and package to curves on parametric surfaces. We describe a theoretical framework and survey the implementation. In particular, we explain how the adaptation to different surfaces and curves is encapsulated in two actual components, namely, the geometry-traits and the topology-traits concepts. We introduce both in Section 4. Section 5 surveys already existing concretizations. We finally give some concluding remarks and future-work directions in Section 6.

## 2. The Basic Arrangement Framework

In the `Arrangement_2` framework of CGAL the combinatorial, graph-like structure (the topology) is separated from the actual embedding on the surface (the geometry). The separation between the topological and the geometric aspects of the subdivision is a key principle not only of the arrangement framework but also of many other CGAL components, such as the various triangulation schemes, computations of Voronoi diagrams, and convex-hull algorithms; see [1] for more details. This separation is enabled by a modular design, and conveniently implemented within the *generic-programming* paradigm. In our extension, this separation becomes even more evident.

### 2.1. The `Arrangement_on_surface_2` Class Template

Our novel framework is implemented as a class template `Arrangement_on_surface_2` parameterized by template parameters *geometry traits* and *topology traits*, that is,

```
Arrangement_on_surface_2< GeoTraits, TopTraits >.
```

A concretization is obtained by instantiating the class template, where models of the geometry-traits and the topology-traits concepts substitute the two template parameters, respectively. The geometry-traits class introduces the C++ type names of the basic geometric objects (i. e., point, curve, and monotone curve) and a small set of operations on objects of these types, such as comparing two points in *xy*-lexicographic order and computing intersections of curves; see Section 4 for the full specification of this concept. The topology-traits class deals with the topology of the surface; see Appendix A for details. In particular, it maintains a representation of the arrangement graph in the form of an *extended doubly-connected edge list* (EDCEL) data-structure as is suitable for the particular topology. An EDCEL is a DCEL [13, Section 2.2] with additional storage (see Section 3.2.4) for a topologically consistent representation.

## 2.2. The Bentley-Ottmann Sweep

Bentley and Ottmann [5] introduced the sweep algorithm for computing intersections of line segments. It is usually formulated for inputs in general position. Already the original paper states that it applies to general  $x$ -monotone curves. The algorithm sweeps the plane with a vertical line starting from  $x = -\infty$  toward  $x = +\infty$ , while maintaining the set of curves intersecting this line. These curves are ordered according to the  $y$ -coordinate of their intersection with the vertical line and stored in a balanced search tree, called the *status structure*. Its content changes only at a finite number of *events*, where an event corresponds to a curve's endpoint or to an intersection between curves. The events are processed in ascending  $xy$ -lexicographic order and stored in an *event queue*. This queue is initialized with the endpoints of all curves. At an event, (i) new curves are added to, and swept curves are removed from, the status structure, (ii) curves swap their positions in the status structure, and (iii) newly adjacent curves are checked for intersections to the right of the sweep line.<sup>3</sup> Any such intersection is inserted into the event queue. The CGAL implementation of the sweep-line algorithm handles all degeneracies, such as multiple curves intersecting in the same point, overlapping curves, or co-vertical events. It is based on the implementation described in [27].

The *canonical output* of the sweep-line algorithm consists of the events in lexicographic order along with adjacency information, that is, which events are connected by a (sub)curve. The CGAL implementation decouples the “bare sweep” procedure from the construction of the actual output using the *visitor* design pattern [21]. Examples of visitors [32] are: A visitor that reports all intersections, a visitor that converts the canonical output into an EDCEL representing the arrangement, a visitor that inserts a set of curves into an existing arrangement, a visitor that overlays two arrangements, a visitor that performs batched point-location, or a visitor that reports the vertical decomposition of the arrangement; see, e. g., [22]. Users may introduce their own sweep-based algorithms by implementing an appropriate visitor class.

The actual sweep is preceded by a preprocessing phase (provided by the geometry-traits class) that subdivides input curves into  $x$ -monotone subcurves and isolated points and ensures further conditions that might be required by other operations of the geometry-traits class.

## 2.3. The Zone Traversal

The sweep-line algorithm is mainly used to create a new arrangement from a set of input curves, but it can also be used for inserting a set of input curves into an arrangement. A different approach is to insert the input curves one by one into a growing arrangement traversing the *zone* of the new curve in the arrangement. The zone [22] of an  $x$ -monotone curve  $C$  in an arrangement is the set of cells intersected by it.

---

<sup>3</sup>We say that a geometric object is to the right (resp. left) of another object, if the former has larger (resp. smaller) lexicographic coordinates than the latter.

The zone of a curve  $C$  is computed by locating the left endpoint of  $C$  in the arrangement ( $l$  in the figure to the right), and then “walking” along the curve towards its right endpoint, keeping track of the vertices, edges, and faces crossed on the way; see, e.g., [13, Section 8.3] for the computation of the zone of a line in an arrangement of lines. The zone-traversal algorithm relies on the same geometric primitives as the sweep-line algorithm. It also produces a *canonical* output, which can be further processed by an appropriate visitor. Again, a variety of zone-related visitors is available, for instance, a visitor that only lists the zone of a curve and a visitor that inserts the curve into a given arrangement.

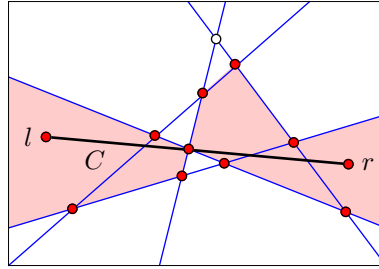


Figure 1: The zone of a segment  $C$  in an arrangement of lines.

### 3. Sweeping and Zoning Over Surfaces

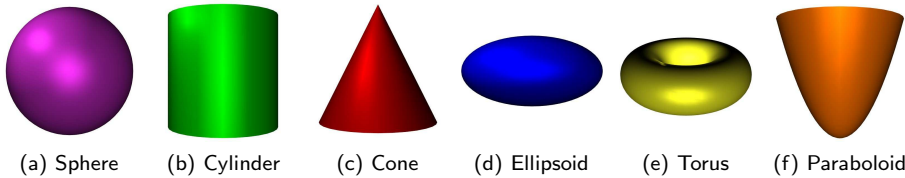


Figure 2: Various two-dimensional parametric-surfaces, arrangements on which are supported by the new framework.

We generalize the framework from the plane to parametric surfaces such as half-planes, cylinders, tori, etc., and surfaces homeomorphic to these; see Figure 2. We aim for an implementation that maximizes code reuse. We mainly discuss the sweep-line algorithm because it is more complex than the zone algorithm. In Section 3.1, we define the class of permissible surfaces and curves, and in Section 3.2, we show how to generalize the algorithms to them.

#### 3.1. Parametric Surfaces and Curves Embedded into Them

We use  $\overline{\mathbb{R}}$  to denote the compactified real line  $\mathbb{R} \cup \{-\infty, +\infty\}$ . The mapping  $x \mapsto x/(1-x^2)$  is a homeomorphism between  $(-1, +1)$  and  $\mathbb{R}$  and between  $[-1, +1]$  and  $\overline{\mathbb{R}}$ . So the reader may also think of finite intervals instead of the (compactified) real line in what follows. We next define details on a parametric surface required to formally describe the framework.

**Definition 3.1 (Parametric Surface).** A parametric surface  $S$  is given by a continuous function  $\phi_S : \Phi \rightarrow \mathbb{R}^3$ , where the domain  $\Phi = U \times V$  is a rectangular two-dimensional parameter space;  $S = \phi_S(\Phi)$ .  $U$  and  $V$  are open, half-open, or closed intervals with endpoints in  $\overline{\mathbb{R}}$ . We use  $u_{\min}$ ,  $u_{\max}$ ,  $v_{\min}$ , and  $v_{\max}$  to denote the endpoints of  $U$  and  $V$ , respectively.

- The left side of the boundary of  $\Phi$  consists of the points  $(u_{\min}, v)$  with  $v \in V$ . It is *open*, if  $u_{\min} \notin U$ , and *closed*, otherwise. The right side is defined analogously. The bottom side consists of the points  $(u, v_{\min})$  with  $u \in (u_{\min}, u_{\max})$ ; the top side is defined analogously. Bottom and top side can also be open. Note that the “corners” of the parameter space belong to the vertical sides; this asymmetry corresponds to the fact, that we sweep  $u$ -monotone curves, and not  $v$ -monotone curves. The four sides together form the boundary  $\partial\Phi$ .
- A point  $p \in S$  is *regular* if it has only one pre-image. All pre-images of a non-regular point lie in the boundary of  $\Phi$ , in particular,  $\phi_S$  is bijective on  $(u_{\min}, u_{\max}) \times (v_{\min}, v_{\max})$ . Moreover, a non-regular point has either exactly two pre-images and then these pre-images lie on opposite sides of the domain or all points of exactly one side of the domain are mapped to it; see Definitions 3.2 and 3.3.

Rectangles, strips, quadrants, half-planes, and planes can be modeled with  $\phi_S$  being the identity mapping. For example,  $\Phi_S(u, v) = (u, v, 0)$  with  $U = V = (-\infty, +\infty)$  parameterizes the plane. Surfaces such as paraboloids can be modeled through continuous and bijective parameterizations, for example,  $\Phi_S(u, v) = (u, v, u^2 + v^2)$ , where  $U = V = (-\infty, +\infty)$ , defines a paraboloid of revolution. Cylinders, tori, spheres, and surfaces homeomorphic to them, require more general parameterizations. For example, the unit sphere is commonly parameterized as  $\phi_S(u, v) = (\cos u \cos v, \sin u \cos v, \sin v)$ , where  $\Phi = [-\pi, \pi] \times [-\frac{\pi}{2}, \frac{\pi}{2}]$ . With respect to this parameterization, the north and the south pole and all points on the opposite Prime (Greenwich) Meridian are non-regular. The north pole  $(0, 0, 1)$  has infinitely many pre-images  $(u, \pi/2)$  with  $-\pi < u < \pi$  and so does the south pole  $(0, 0, -1)$ . The points on the opposite Prime Meridian have two pre-images each, namely  $(-\pi, v)$  and  $(\pi, v)$  with  $-\pi/2 \leq v \leq \pi/2$ . We say that the upper and lower side of the domain are *contracted* and the left and right sides are *identified*. These are exactly the kinds of non-injectivity that we allow.

**Definition 3.2 (Contraction).** A side of the domain is contracted, if  $\phi_S$  is constant on it. The image of the side is called a *contraction point*.

**Definition 3.3 (Identification).** The bottom and top sides of the domain (similarly for the left and right sides) are *identified*, if  $\phi_S(u, v_{\min}) = \phi_S(u, v_{\max})$  for all  $u \in U$ . The curve  $u \mapsto \phi_S(u, v_{\min})$  is called an *identification curve*.

We give more examples. A *triangle* with corners  $(a_1, b_1)$ ,  $(a_2, b_2)$ , and  $(a_3, b_3)$  can be parameterized via  $\Phi = [0, 1] \times [0, 1]$  with  $\phi_S(u, v) = (a_1 + u(a_2 - a_1) + uv(a_3 - a_2), b_1 + u(b_2 - b_1) + uv(b_3 - b_2), 0)$ . The left side of the rectangular domain contracts to a point. An open or closed *cylinder* is modelled by identifying

the vertical sides and having  $V$  open or closed, respectively. A *torus* is modelled by identifying the vertical sides and the horizontal sides. A *paraboloid* or *half-cone* may be modelled by identifying the vertical sides and contracting one of the horizontal sides to a point. More elegantly, they are modelled by a bijective parameterization as given above. A *sphere* is modelled by identifying the vertical sides and contracting both horizontal sides. A *croissant*, a torus with one pinch point, is modelled by identifying the vertical and identifying the horizontal sides and, in addition, contracting one of the pairs. However, the croissant is excluded by our definitions. All surfaces supported by our framework are locally homeomorphic to a disk, and hence an EDCEL data-structure suffices for representing arrangements on these surfaces. The croissant is, at the pinch point, not locally homeomorphic to a disk, and hence a more general cell-tuple data structure than an EDCEL would be needed [11].

We next turn to curves on  $S$ . As usual, a curve is a continuous mapping from a one-dimensional domain. We use the open, half-open, or closed unit interval as the one-dimensional domain, that is, a “curve-end” may or may not belong to the curve. If a curve-end does not belong to the curve, the pre-image of the curve must approach an open side of  $\Phi$ . Curves must have only a finite number of self-intersections.

**Definition 3.4 (Curve).** A *parameterizable curve*  $\gamma$  is a continuous function  $\gamma : I \rightarrow \Phi$ , where  $I$  is an open, half-open, or closed interval with endpoints 0 and 1, and  $\gamma$  is injective except for at a finite number of points. If  $0 \notin I$ ,  $\lim_{t \rightarrow 0+} \gamma(t)$  exists (in the closure of  $\Phi$ ) and lies in an open side of the boundary. Similarly, if  $1 \notin I$ ,  $\lim_{t \rightarrow 1-} \gamma(t)$  exists and lies in an open side of the boundary. A curve  $C$  in  $S$  is the image of a curve  $\gamma$  in the domain.

A curve is *closed in the domain* if  $\gamma(0) = \gamma(1)$ ; in particular,  $0 \in I$  and  $1 \in I$ . A curve is *closed in the surface  $S$  (or simply closed)* if  $\phi_S(\gamma(0)) = \phi_S(\gamma(1))$ . A curve  $\gamma$  has two *ends*, the 0-end  $\langle \gamma, 0 \rangle$  and the 1-end  $\langle \gamma, 1 \rangle$ . If  $d \in I$ , the  $d$ -end has a geometric interpretation. It is a point in  $\Phi$ . If  $d \notin I$ , the  $d$ -end has no geometric interpretation. You may think of it as a point on an open side of the domain or an initial or terminal segment of  $\gamma$ . If  $d \notin I$ , we say that the  $d$ -end of the curve is open. The equator curve on the sphere in standard parameterization is given by  $\gamma(t) = (\pi(2t - 1), 0)$  for  $t \in [0, 1]$ . The 0-end of  $\gamma$  is the point  $(-\pi, 0)$  in  $\Phi$  and a point on the equator of the sphere. It is closed on the sphere, but non-closed in  $\Phi$ . The diagonal  $(u, u)$  in the plane is, for example, given by  $\gamma(t) = (u(t), v(t))$  and  $u(t) = v(t) = (t - 1/2)/(t(1 - t))$ . Both ends of this curve are open. The  $d$ -end of a curve  $\gamma$  is incident to the left side if either  $d \in I$  and  $\gamma(d)$  lies on the left side or  $d \notin I$  and  $\lim_{t \rightarrow d} \gamma(t)$  lies on the left side, which is then an open side. Similarly for the other sides.

**Definition 3.5 ( $u$ -monotone curve).** A *strongly  $u$ -monotone curve* is the image of a curve  $\gamma$ , such that if  $t_1 < t_2$ , then  $u(\gamma(t_1)) < u(\gamma(t_2))$ . A *vertical curve* is the image of a curve  $\gamma$ , such that  $u(\gamma(t)) = c$  for all  $t \in I$  and some  $c \in U$

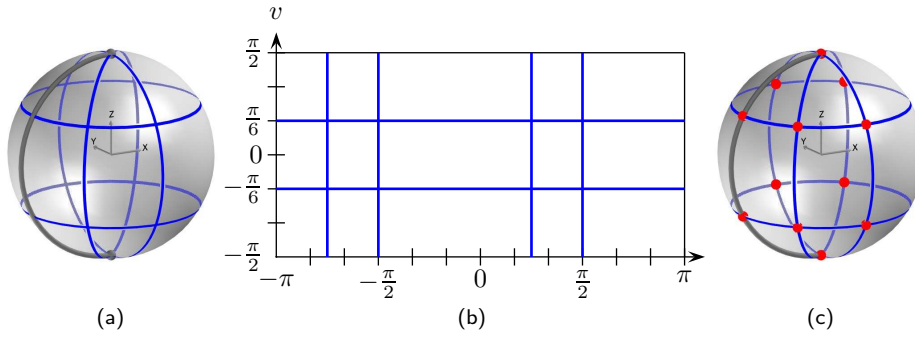


Figure 3: (a) A sphere with two great circles passing through the poles and two circles parallel to the equator. The identification curve is shown in grey. (b) The pre-images of the curves in the parameter space. (c) The  $\text{ED}_{\text{CEL}}$ . Observe the two vertices on the identification curve.

and  $v(\gamma(t_1)) < v(\gamma(t_2))$  for  $t_1 < t_2$ . For instance, every Meridian curve of a sphere parameterized as above is vertical. A  $u$ -monotone curve is either vertical or strongly  $u$ -monotone.

**Definition 3.6 (Sweepable curve).** A curve is *sweepable* if it is  $u$ -monotone and does not touch the boundary in its interior, that is,  $\gamma(t) \in \Phi \setminus \partial\Phi$  for all  $t \in (0, 1)$ .

We can now formally state the goal of our work: *Compute the arrangement induced by a set of curves on a surface. The surface must be parameterizable as defined above, and the curves must be decomposable into parameterizable subcurves. Any two curves in the set intersect only in a finite number of points and overlap only in a finite number of sections.*<sup>4</sup> We need our curves to be *nice*. We do not define this notion formally, but only state that, they must be decomposable into a finite number of parameterizable sweepable subcurves, such that all geometric operations defined in Section 4 can be defined for the subcurves. For instance, algebraic curves are nice, while curves based on trigonometric functions are often not. All input curves are decomposed into a finite number of parameterizable sweepable subcurves in a preprocessing step; see also Section 3.2.2.

### 3.2. The Generalization of the Algorithms

The standard sweep-line algorithm sweeps the plane containing bounded curves. *How do we sweep a surface and process the curves embedded on it?* We (conceptually) sweep the parameter space  $\Phi$  with a vertical line  $\ell$  from  $u_{\min}$  to  $u_{\max}$ . The sweep of  $\Phi$  induces a sweep of the surface through the parameterization  $\phi_S$ . At any time of the sweep,  $\phi_S(\ell)$  is a curve in  $S$ . This curve sweeps  $S$ . The advantage of formulating the sweep for parameter space is that we are on well-known grounds. For example, we have the familiar lexicographic order of points, and we process events in lexicographic increasing order. However, we also need to take care of

<sup>4</sup>In this paper, we do not discuss overlap between curves. The implementation handles them.

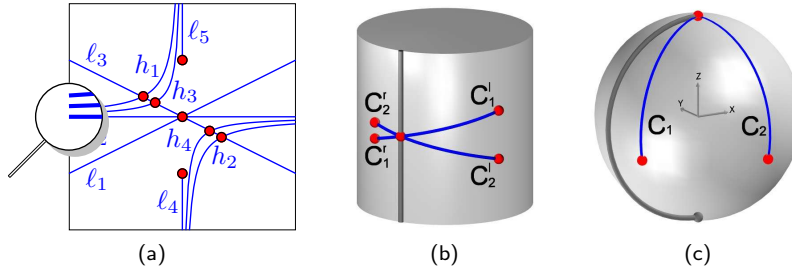


Figure 4: Comparing sweep events. (a) The order of the events is the lexicographic order they appear in the drawing. For example, the maximal end of  $h_3$  is smaller than the minimal end of  $l_4$ , which is smaller than both ends of  $l_5$ . (b) Comparing near the identification curve:  $C_2^l < C_1^l$  to the right of identification curve. (c) Comparing near a point of contraction: maximal end of vertical  $C_1$  is smaller than maximal end of vertical  $C_2$ .

additional issues; see Figure 3 for an illustration of the second and third of the following items:

- (1) A curve-end may or may not correspond to a point on  $S$ . Since a curve-end is an event, we need a more general notion of event. We also need to extend the notion of lexicographic order.
- (2) The surface may have non-regular points. Such points have more than one pre-image, and thus are swept more than once in parameter space.
- (3) Curves incident to a non-regular point are discovered at unrelated events. Our framework has to correlate these events and make the appropriate contractions or identifications.
- (4) We use the language of parameterization in our arguments and definitions. *We do not assume that either the surface or the input curves are given through their parameterization*, and hence the implementations of the geometry-traits and topology-traits classes usually work entirely over  $S$ . The specific parameterization is reflected by the operational semantics of the traits classes.

In the following, we describe how these issues are addressed.

**3.2.1. Events.** The events in the standard sweep are endpoints, intersection points, and isolated points. We generalize endpoints to curve-ends, and so our events are now curve-ends, intersection points, and isolated points. A curve-end corresponds either to a point in  $S$  or is open. In the latter case, the respective part of the curve approaches a side of the domain. An intersection point in the interior of a curve is necessarily regular, as we sweep only sweepable curves. We define the lexicographic ordering on events in Section 3.2.3.

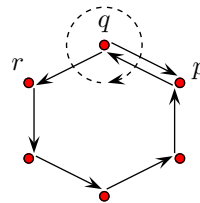
**3.2.2. Non-Injectivity on the Boundary.** Points of contraction and points on an identification curve have multiple pre-images. Instead of sweeping over the entire parameter space  $\Phi$ , we sweep over its interior  $\Phi \setminus \partial\Phi$ , or alternatively viewed,

over the modified surface  $\tilde{S} = \phi_S(\Phi \setminus \partial\Phi)$ . This way our algorithm handles only  $u$ -monotone curves, the interior of which is disjoint from the boundaries of  $\Phi$ . Isolated points and curves that lie in  $\partial\Phi$  are handled separately. In the preprocessing stage we split the input curves into sweepable subcurves. In the example shown in Figure 4(b), the curves  $C_1$  and  $C_2$  cross the curve of identification. Their pre-images in parameter space are curves  $\gamma_1$  and  $\gamma_2$  having their 0-ends on the left side of  $\partial\Phi$  and their 1-ends on the right side of  $\partial\Phi$ . A curve in  $S$  that winds around the cylinder several times gives rise to several curves in parameter space that extend from the left side to the right side. For each sweepable curve, we have the 0-end and the 1-end events. We remark that breaking curves into sweepable curves may result in additional vertices in the EDCEL. These vertices are induced by the chosen parameterization of  $S$  rather than by the original input curves.

**3.2.3. Comparing Events.** Events in the standard sweep-line procedure are associated with points. Now curve-ends are also events. We extend the geometric operations used by the standard sweep, and add new ones, to accomplish the sweeping task. These operations, like all other geometric operations, are provided by the geometry-traits class. *How are curve-ends and points compared?* We distinguish cases, many of which are handled in a straightforward manner. For example, it is clear that the curve-end of a curve approaching the left side is smaller than any point lying in the interior of the parameter space, which in turn is smaller than the curve-end of any curve approaching the right side. We compare two curve-ends of curves approaching the left side by considering their intersections with a vertical line at  $u = u_{\min} + \varepsilon, \varepsilon > 0$ , and return the  $v$ -order of these points.  $\varepsilon$  is sufficiently small, such that the result does not depend on the choice of  $\varepsilon$ . Similar rules apply to the other situations; see Figure 4 for illustrations and Section 4 for details.

The sweep now proceeds almost unchanged. We initialize the event queue with the events for curve-ends and isolated points. However, instead of starting to process the top event in the queue and proceeding one at a time, we optimize and first consider the initial sequence of the event queue that consists of curve-ends of curves approaching the left side. We insert their incident curves into the status structure in the order of their corresponding events in the event queue, avoiding further geometric comparisons, which are expensive especially when high-degree curves are involved. Similarly, towards the end of the process, when the event queue contains only curve-ends ending in the right side, we remove in one blow all these events from the event queue and their incident curves from the status structure.

**3.2.4. Constructing and Maintaining the EDCEL.** The DCEL is a popular data structure for representing graphs embedded into an orientable surface. For such an embedded graph, the edges incident to any vertex are sorted in clockwise order around the vertex. Two vertices  $p$  and  $q$  are linked by twin halfedges  $(p, q)$  and  $(q, p)$  that are oriented opposite to each other. Each halfedge has a successor; the successor of the



halfedge  $(p, q)$  is the halfedge  $(q, r)$ , where  $(q, r)$  is the edge following  $(q, p)$  in the clockwise ordering of edges around  $q$ . The vertices and halfedges of the DCEL form a directed graph. Two vertices belong to the same (connected) component of this graph, if they are connected by an undirected path. Isolated vertices form trivial components. Any non-trivial component decomposes into cycles of halfedges formed by the successor relation. The halfedges in each such cycle have the same face to their left. This face is stored with each halfedge record. The boundary of each face consists of a number of such cycles; we call them the *connected components of the boundaries* (CCBs) of the face. A face record stores a pointer to one halfedge of each of its CCBs. The CCBs contained in any component of the DCEL contribute to the boundaries of the faces incident to the component.

The topology-traits class knows about the topology of the surface, that is, for each side of the surface parameter-space, whether it is open, closed, contracted, or identified. It maintains the representation of the arrangement as an EDCEL. An EDCEL differs from a DCEL in two aspects. First, it allows to maintain fictitious vertices and fictitious edges, that is, records not storing geometric points or curves, respectively. Second, while a DCEL-face stores a single outer CAB and a (possibly empty) set of inner CCBs for each face, an EDCEL-face maintains besides the set for inner CCBs also (possibly empty) set of outer CCBs. We learn in this section why those modification make sense for representing arrangements induced by curves on surfaces with special boundaries.

The algorithms and the topology-traits class communicate with each other through methods provided by the latter. For example, consider two sweep events that are associated with the same (identified or contracted) point in  $S$ . The topology-traits class establishes the association, and informs the sweep algorithm, which is otherwise not aware of this effect.

Events taking place in the interior of  $\Phi$  bring nothing new. They are handled as usual; appropriate EDCEL records are constructed and properly linked.

**Open Boundaries:** The DCEL data-structure is capable of representing bounded arrangements in the plane and related topological structures. It was not designed to deal with open curve-ends. We have to handle open curve-ends, and we would like to do this with as little dedicated treatment as possible. Consider, for instance, Voronoi diagrams of points in the plane represented as arrangements. Such an arrangement has rays approaching infinity that must be properly handled. In addition, the traversal of the boundary of a face should not depend on whether the face is bounded or unbounded.

We first describe two solutions for the plane and then comment on other surfaces. For the plane, one solution is to add a single vertex at infinity, and the other solution is to add a cycle at infinity. Other consistent solutions exist, but these two solutions have simple geometric interpretations.

The first solution (*single vertex at infinity*) corresponds to viewing the plane as the image of a punctured sphere (namely, a sphere with the north pole removed) under stereographic projection. The north pole itself gives rise to a single vertex at

infinity, say  $V_{\text{inf}}$ . All unbounded curves are incident to it, and the cyclic ordering of these curves around  $V_{\text{inf}}$  is well-defined. So in our EDCEL, any unbounded face is incident to  $V_{\text{inf}}$ . The traversal algorithms for faces work essentially without change. The only change is that they must report whether a traversed vertex is real or fictitious,  $V_{\text{inf}}$  being fictitious.

The second solution (*implicit bounding cycle at infinity*) corresponds to viewing the plane as the image of a lower hemisphere under projection from the center. The equator maps to a circle at infinity. For technical reasons, we prefer a rectangle at infinity. There are fictitious vertices corresponding to the corners of the implicit rectangle and one for each curve-end at infinity. The fictitious vertices are linked into a cycle by fictitious edges. The insertion of an open curve requires splitting a fictitious edge. Traversals must filter out all fictitious vertices and edges. In this representation, there is also a fictitious face, the face outside the implicit bounding rectangle. It stands for the upper hemisphere in the projection from the center of the sphere. The face traversal must filter out this face.

The topology-traits class decides for each open side how to represent curves approaching it, and whether to have a joint representation for neighboring (open) sides. We remark that our solution for open sides also applies to *closed* sides. In particular, there is no need for the user to add an artificial curve that exists on the image of the closed boundary-side. In fact, we prefer a fictitious closure, as this enables the possibility to have a face incident to the boundary side that is “open” on that closed side of the parameter space (i. e., not having an actual curve on that side).

**Contractions and Identifications:** Consider a sweep of the sphere with some circles embedded into it; see Figure 3. In parameter space, a great circle passing through the poles is a pair of vertical segments having their curve-ends on the lower and upper sides, and a circle parallel to the equator is a horizontal line having its curve-ends on the left and right sides. So in the sweep of the parameter space, we would have several copies of the north pole and south pole, and the circles parallel to the equator would appear as non-closed curves. In the EDCEL, we want only one copy of each pole and the circles parallel to the equator to be represented as a closed sequence of edges. The topology-traits class makes the required contractions and identifications.

We describe the mechanism for identification curves, points of contraction being simpler. A topology traits maintains a sorted sequence of EDCEL vertices for each identification curve. Assume for concreteness that the left and right sides are identified. The first vertex associated with an identified point at a certain  $v$ -value is created as usual, and also inserted into the sorted sequence. When this vertex is to be created for the second time because its second pre-image is encountered by the sweep, the topology-traits class detects that the vertex already exists, and hence does not create a new vertex. Rather, the already existing vertex is used instead. In this way, the proper identifications are made.

The discussion above readily extends to other surfaces. For example, in the case of a cylinder, the EDCEL would have (besides the identification) two fictitious

circles — one for the upper rim of the cylinder and one for the lower rim; see Figure 4(b). In the case of the paraboloid  $z = x^2 + y^2$ , the EDCEL would have (besides the identification) one circle at infinity. The option to use a single fictitious vertex instead of a fictitious circles is also conceivable.

**Face Types, Nesting, Inner and Outer CCBs:** The faces in an arrangement have different homeomorphism types. For the surfaces considered in this paper, we have faces homeomorphic to punctured disks (disk-like faces), punctured cylinders (cylinder-like faces), punctured spheres (sphere-like faces), and punctured tori (torus-like faces). In the plane, there is a natural notion of nesting of faces. We extend this notion to all surfaces under consideration. Nesting is exploited in a number of algorithms, for example point location and face traversal. We show below how the knowledge of face types and nesting simplifies the update step after the insertion of a curve into the arrangement.

Recall that the EDCEL graph decomposes into connected components; see Figure 5a. An isolated vertex is a component of its own. A face can be incident to more than one component. The face-component graph has the faces and components as vertices; see Figure 5b. A face and a component are connected in this graph, if they are incident to each other in the arrangement. The face-component graph is connected. We classify each CCB of every face as *outer* or *inner*. This information is encoded in the face-component graph as follows: Let  $F$  denote a face, and let  $K$  denote a component incident to  $F$ . Exactly one CCB of  $F$  is *contained* in  $K$ . We direct the edge from  $K$  to  $F$ , if  $K$  contains an outer CCB of  $F$ , and from  $F$  to  $K$ , if  $K$  contains an inner CCB of  $F$ . We need one more definition before we can proceed. We call the maximal connected subsets of  $S \setminus K$  the *regions* of  $K$ .

We start with the familiar case of bounded curves in the plane. Let  $K$  be some component. According to Jordan's curve theorem, one region of  $K$  is unbounded, say  $R_0$ , and all others, say  $R_1$  to  $R_k$  are bounded. The bounded regions are nested in the unbounded region. We extend nestedness to faces. For each  $K$ , there is a unique face  $F_i$  in the arrangement that is incident to  $K$  and contained in  $R_i$ . Then, the faces  $F_i$ ,  $i \geq 1$  are nested in  $F_0$ . Let  $B_i$  be the CCB of  $F_i$  contained in  $K$ .  $B_0$  is an inner CCB of  $F_0$ , and  $B_i$ ,  $i \geq 1$ , is an outer CCB of  $F_i$ . The face-component graph is a tree, as each face, except for the unbounded one, has a single outer CCB. The unique unbounded face is the root of the tree. The unique faces incident to  $K_1$  in Figure 5 are  $F_0$ ,  $F_1$ , and  $F_2$ .  $F_0$  is contained in the unbounded region of  $K_1$ . Thus,  $F_1$  and  $F_2$  are nested in  $F_0$  (via  $K_1$ ). The unique

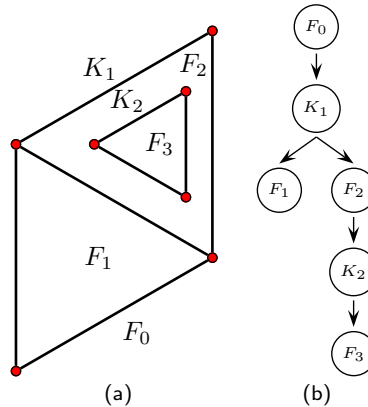


Figure 5: The directed face-component incidence graph of a planar embedded graph induced by bounded curves.

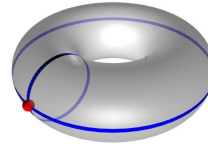
face-component graph is a tree, as each face, except for the unbounded one, has a single outer CCB. The unique unbounded face is the root of the tree. The unique faces incident to  $K_1$  in Figure 5 are  $F_0$ ,  $F_1$ , and  $F_2$ .  $F_0$  is contained in the unbounded region of  $K_1$ . Thus,  $F_1$  and  $F_2$  are nested in  $F_0$  (via  $K_1$ ). The unique

faces incident to  $K_2$  are  $F_2$  and  $F_3$ .  $F_2$  contained in the unbounded region of  $K_2$ . Thus,  $F_3$  is nested in  $K_2$  (via  $K_2$ ).

We first consider an unbounded component in the plane. There are two natural approaches for defining a nesting. The first corresponds to viewing the plane as the stereographic projection of a sphere, which means that we are in the situation discussed in the preceding paragraph. We introduce a single vertex at infinity, and chose one of the unbounded faces as the special face. All other faces are nested in it. The face-incident graph is a tree rooted at the chosen special face. The second approach corresponds to viewing the plane as the projection of the lower hemisphere. We introduce a circle at infinity, which is the projection of the equator. The upper hemisphere projects into a fictitious face outside the circle at infinity. We choose the fictitious face as the special face and again have a nesting. This is the approach we implemented. In our implementation the circle is replaced by an implicit rectangle, which comprise of four fictitious edges. The same approaches are applicable to bounded surfaces homeomorphic to an (open) disc.

On a sphere, there is no natural distinction between the regions incident to a component. However, there is a well-known remedy. We fix an arbitrary reference point (not lying on any edge or vertex) on the sphere, and designate the face containing the reference point to be the “unbounded” special face. This establishes a well-defined nesting.<sup>5</sup> There is another way of defining the nesting. We select an arbitrary component as outermost, declare the CCBs contained in it as outer, and have all other faces and components nested within them.

A closed curve on a surface is called *contractible*, if it can be continuously contracted to a point. On a surface homeomorphic to a disc or sphere, all closed curves are contractible. A cylinder may contain non-contractible curves, and so may the sphere punctured at the poles, which is the way we view the sphere. The torus, being a surface of genus one, may contain *two different types* of non-contractible curves, as depicted in the figure above. Formally, a closed curve  $C$  is contractible if and only if the number of times  $C$  intersects any curve of identification is odd. The curves of identification themselves are non-contractible. The treatment of the cylinder is a special case of the treatment of the torus described below, and the sphere (punctured at the poles) is treated exactly like the cylinder is treated, as the boundary of the parameter space of both surfaces have two identified and two non-identified sides. We call a component *contractible*, if no non-contractible closed curve is contained in it. Only the components  $K_3$  and  $K_4$  in Figure 6 are contractible.



Consider the components of an arrangement graph embedded on the torus. If all components are contractible, all faces of the arrangement except for one

<sup>5</sup>In our implementation we designated the face containing the point  $p = (u_{\max}, v_{\max} - \varepsilon)$ ,  $\varepsilon > 0$  as the unbounded face.  $\varepsilon$  is chosen, such that  $p$  does not lie on any edge and does not coincide with any vertex, and is sufficiently small, such that the nesting does not depend on the choice of  $\varepsilon$ .

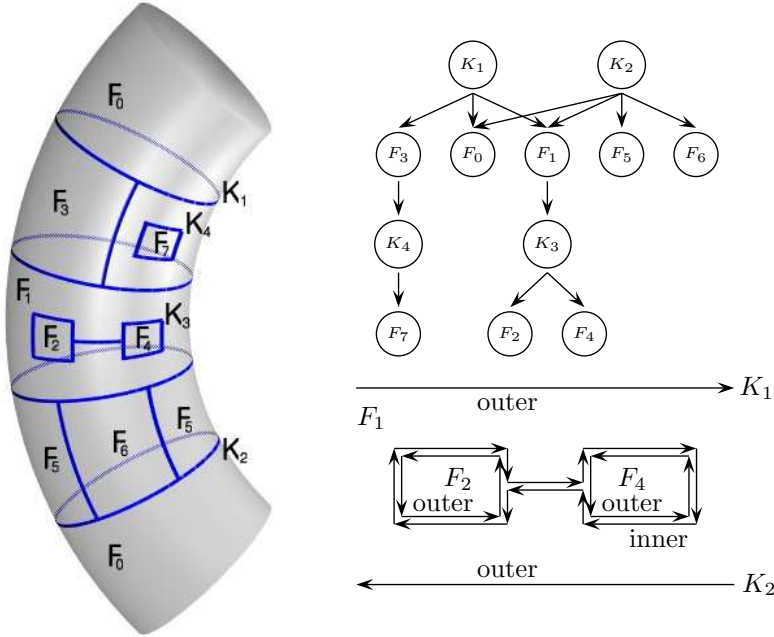


Figure 6: A graph embedded on a (partially displayed) torus and the corresponding directed face-component incidence graph.  $F_0$  and  $F_1$  are cylinder-like, and all other faces are disk-like.  $F_0$  and  $F_1$  have two outer CCBs each, one contained in  $K_1$  and one contained in  $K_2$ ;  $F_1$  also has one inner CCB. The lower drawing on the right shows a close-up view of the CCBs of  $F_1$ ,  $F_2$ , and  $F_4$ .

are punctured disks, and one face, say  $F_0$ , is a punctured torus. The disk-like faces are nested in  $F_0$ , and all CCBs of  $F_0$  are inner. So assume that there are non-contractible components  $K_1, K_2, \dots, K_\ell$ . If some component contains both types of non-contractible closed curves,  $\ell = 1$  and all regions of  $K_1$  are disk-like. The CCBs contained in  $K_1$  are outer CCBs for the corresponding faces. Within each such face, we have the situation of the plane with bounded components. So assume next that no component contains both kinds of non-contractible cycles. Then, all non-contractible components contain the same kind of non-contractible cycles, as instances of different kinds necessarily cross.  $S \setminus (K_1 \cup \dots \cup K_\ell)$  consists of disk-like and cylinder-like regions. A disk-like face is incident to exactly one of the  $K_i$ 's, and a cylinder-like face is incident to two. The unique CCB of a disk-like face contained in one of the non-contractible  $K_i$ 's is outer, and so are the two CCBs of a cylinder-like face contained in two of the non-contractible  $K_i$ 's respectively. The (undirected) face-component graph consists of a cycle containing

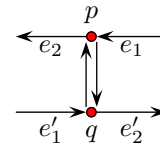
the non-contractible  $K_i$ 's and all cylinder-like faces as well as sub-trees rooted at the nodes of this cycle.  $K_1, F_0, K_2, F_1$  in Figure 6 forms an example of such a cycle.

We conclude this paragraph with some remarks on the face-component graph: In the directed face-component graph, the  $K_i$ 's form the top level, the cylinder-like faces have two parents, and the disk-like faces have one. The directed face-component graph is acyclic, but not necessarily a tree, as each cylinder-like face has two outer CCBs. In the non-directed face-component graph, the cylinder-like faces and the components separating them form a cycle. In case of cylinder, they form a chain.

**Maintaining CCBs:** The addition of an isolated vertex  $p$  creates a trivial component nested in the face containing  $p$ . The addition of an edge  $\{p, q\}$  to the arrangement graph adds halfedges  $(p, q)$  and  $(q, p)$  to the EDCEL. The CCBs are easily updated. The updating of the nesting is more demanding. We only discuss the torus, the other surfaces being simpler.

If  $p$  and  $q$  both have degree one after the addition, we have a new CCB consisting of the two new halfedges. It is nested in some face. If exactly one of  $p$  and  $q$  has degree one after the addition, say  $q$ , we only have to insert  $(p, q)$  into the right position in the cyclic order of edges around  $p$ . This inserts  $(p, q)$  followed by  $(q, p)$  into one of the existing CCBs.

It remains to discuss the case where both  $p$  and  $q$  have degree at least two after the addition. Note that  $p = q$  is impossible, because we insert only  $u$ -monotone curves into the arrangement. We first show how to update the CCBs, and then how to update the face-component graph. Assume that  $(p, q)$  becomes the successor of some halfedge  $e_1$  and  $(q, p)$  becomes the successor of  $e'_1$ , as depicted in the Figure to the right. The old successor, say  $e_2$ , of  $e_1$  becomes the successor of  $(q, p)$  and the old successor, say  $e'_2$ , of  $e'_1$ , becomes the successor of  $(p, q)$ . After the addition of  $\{p, q\}$ , we have  $e_1 \rightarrow (p, q) \rightarrow e'_2$  and  $e'_1 \rightarrow (q, p) \rightarrow e_2$  as part of CCBs.



For the update of the face-component graph, we distinguish cases according to whether the addition of  $\{p, q\}$  increases or decreases the number of CCBs.

Assume first that the number of CCBs increases. This is the case when  $e_1$  and  $e'_1$  exist and belong to the same CCB before the addition. Let  $F$  be the face to the left of  $e_1$ . The new halfedges  $(p, q)$  and  $(q, p)$  are added to the component that contained  $e_1$ ; call it  $K$ . We need to know whether  $K$  becomes non-contractible by the addition. So assume that  $K$  was contractible before the addition. We consider any closed cycle in the altered  $K$  containing  $\{p, q\}$ . If the cycle intersects the identification curves an odd number of times, we have created a non-contractible component. Otherwise, we have not. Observe, that this decision is independent of how the closed cycle is formed, because any closed cycle in the component before the addition of  $\{p, q\}$  had an even number of intersections with the identification curves. We distinguish cases according to whether  $K$  is (i) contractible before and

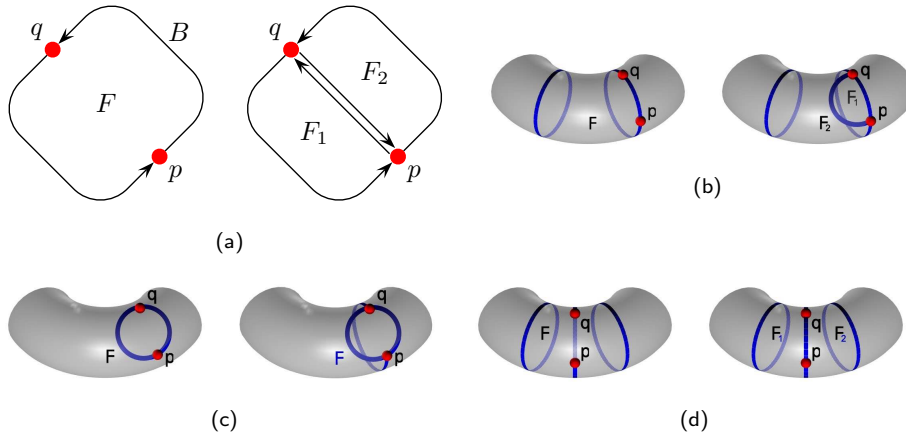


Figure 7: The number of CCBs increases by the addition of  $\{p, q\}$ . (a) and (b) illustrate the case where the type of the component containing  $p$  and  $q$  does not change. In (b), a cylinder-like face  $F$  is split into a cylinder-like  $F_2$  and a disk-like face  $F_1$ . In (c), a torus-like face is converted into a cylinder-like face, and in (d), a cylinder-like face is split into two cylinder-like faces.

after the addition, (ii) contractible before and non-contractible after the addition, or (iii) non-contractible before and after the addition.

We start with cases (i) and (iii). The addition splits a face  $F$  into two faces  $F_1$  and  $F_2$ , one being disk-like, the other having the same type as  $F$ . For the disk-like face we create a new outer CCB. For the face inheriting the type of  $F$ , the type (i. e., inner or outer) of the CCB is also inherited. But we have to assign the new CCBs to  $F_1$  and  $F_2$ :

- If  $F$  was disk-like, the assignment is as in the plane; see Figure 7a
- If  $F$  was cylinder-like and we split one of its outer CCBs, one of the resulting CCBs is contractible and one is non-contractible. The non-contractible CCB is assigned to the cylinder-like face and the other one is assigned to the disk-like face; see Figure 7b
- If  $F$  was cylinder-like and we split one of its inner CCBs, we have to distinguish whether  $K$  is (and stays) contractible or whether  $K$  is (and stays) non-contractible. In the former case, the inner CCB can be considered as embedded in a disk-like face, and we process accordingly. In the latter case, the situation is analogue to splitting an outer CCB in a cylinder-like face.

If the component becomes non-contractible, as in (ii), a torus-like  $F$  becomes cylinder-like (see Figure 7c) or a cylinder-like  $F$  is split into two cylinder-like faces  $F_1$  and  $F_2$  (see Figure 7d). In the former case, both new CCBs are outer CCBs of the now cylinder-like  $F$ . In the latter case, the two outer CCBs of  $F$  plus the two

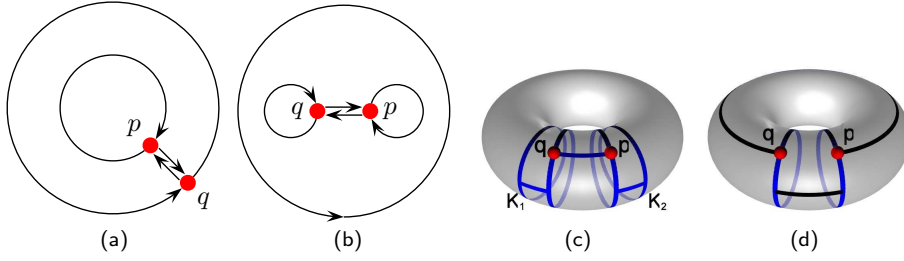


Figure 8: Two CCBs are joined into one by the addition of  $\{p, q\}$ . Four cases can arise. We either join (a) an outer and an inner CCB, or (b) two inner CCBs, or two outer CCBs: In the latter case, the two outer CCBs either belong to (c) distinct components, or (d) to the same component.

new CCBs become the outer CCBs of  $F_1$  and  $F_2$ . How do we assign the two new ones to  $F_1$  and  $F_2$ ? Let  $B$  and  $B'$  be the outer CCBs of  $F$ . Both of them have an odd number of intersections with the identification curves. Since they have the same type, there is one identification curve that is crossed an odd number of times by both. For ease of exposition, assume that this identification curve corresponds to the vertical sides. Assume that  $B$  crosses the identification curve more often from left to right than from right to left, say  $c$  times more often. Then  $B'$  has  $c$  more crossings from right to left. We simply count the number of crossings for each one of the new CCBs. One must have  $c$  more crossing from left to right and the other  $c$  more crossings from right to left.  $B'$  becomes the second outer ccb's of the  $F_i$  incident to the former, while  $B$  is paired with the latter to form the outer CCBs of the other face.

In all cases we also have to redistribute the inner CCBs (and isolated points) of  $F$  over  $F_1$  and  $F_2$ . For this it suffices to locate one point of each CCB with respect to  $F_1$  and  $F_2$ .

Assume next that  $e_1$  and  $e'_1$  belong to distinct CCBs before the addition, say  $B$  and  $B'$ , respectively; see Figure 8. Let  $K$  and  $K'$  be the components containing them;  $K = K'$  is possible. The two CCBs are joined into one, so the number of CCBs decrease. Let  $F$  be the face to the left of  $e_1$  and  $e'_1$ . We distinguish cases and see that the case distinction can be made knowing the kinds (inner or outer) of  $B$  and  $B'$  and whether  $K \neq K'$  or not. If  $B$  is an outer CCB and  $B'$  is an inner CCB of  $F$  (or vice versa), the new edge joins two components that were nested within each other before the addition. The joined CCB is an outer CCB of  $F$ ; see Figure 8a. If  $B$  and  $B'$  are inner CCBs of  $F$ , two components nested in  $F$  join, and the joined CCB is an inner CCB of  $F$ ; see Figure 8b. If  $B$  and  $B'$  are outer CCBs of  $F$  and  $K \neq K'$ , the cylinder-like face  $F$  turns into a disk-like face. The joined CCB is an outer CCB of  $F$ ; see Figure 8c. If  $B$  and  $B'$  are outer CCBs of  $F$  and  $K = K'$ ,  $F$  is the only cylinder-like face. It turns into a disk-like face and

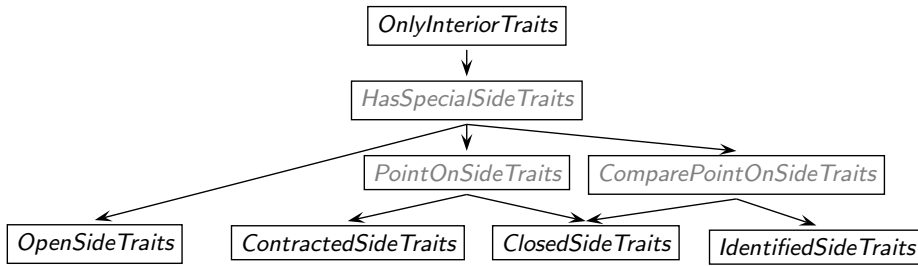


Figure 9: Top portion of the refinement hierarchy of the geometry-traits concepts. The grey concepts factor out operations that are common to their descendants. The black concepts correspond to application scenarios. There are four copies of the concepts at the bottom — one for each side; see also Figure 10 and 11.

now all faces are disk-like. The joined CCB is an outer CCB of  $F$ ; see Figure 8d. Similar considerations are required when an edge is deleted from an arrangement.

**3.2.5. Inserting a curve via zone traversal.** Recall that inserting a curve using zone traversal requires as first step locating the EDCEL-record (i. e., face, edge, or vertex) that represents the minimal end of the given curve. In case of bounded curves in the plane, this is achieved by a *point-location* query. Now, the minimal end might be an open end or lie on a side of the domain. So we postulate a specific point-location strategy for these situations. The rest of the algorithm is carried out using the same geometric and topological operations needed by the sweep-line algorithm.

## 4. The Geometry Traits and Topology-Traits Concepts

The topology-traits concept and the geometry-traits concepts and their refinement hierarchy are defined according to the identified demands imposed by different algorithms that operate on arrangements induced by curves embedded on a surface. We believe that the requirements listed by the concepts include only the utterly essential types and operations, and fully specify all the preconditions that the input must satisfy, as these may simplify the implementation of models of these concepts.

Section 4.1 describes in detail the basic concept of the geometry-traits hierarchy *OnlyInteriorTraits*, models of which are sufficient to deal with bounded curves in the plane. Models of the refined concepts in the hierarchy (see Figure 9) can handle more complicated situations such as curves approaching open or reaching closed, contracted, or identified sides. The details of the refined concepts are given in Section 4.2. Section 4.3 contains an overview of the topology traits.

#### 4.1. The Basic Geometry-Traits Concept

The root concept *OnlyInteriorTraits* matches the original *ArrangementTraits\_2* concept; it is sufficient for constructing and manipulating arrangements of planar bounded curves in  $U = V = (-\infty, +\infty)$ , and homeomorphic situations. It requires the definition of the three types: general curve,  $u$ -monotone curve, and point. General curves are used only as input curves. They are not necessarily  $u$ -monotone, may comprise several disconnected branches, or contain self-intersections. For instance, the polynomial  $(u^2 + v^2)(u^2 + v^2 - 1) = 0$  induces an algebraic curve comprising two  $u$ -monotone circular arcs, which together form the unit circle, and a singular isolated point at the origin. The concept contains an operation that subdivides any general curve into a finite number of sweepable curves and isolated points. All further operations, presented below, involve only such curves and also points. Those operations serve as the basis for our generalization.

The functions  $\text{cmp}_u()$  and  $\text{cmp}_v()$  accept pairs of regular points in  $S$  and compare them by their  $u$ -coordinate and by their  $v$ -coordinate, respectively. In the original concept, a point  $p$  is simply a pair  $(u, v)$ , and hence the implementation can be direct. Now, they accept regular points in  $S$ . Such points have unique pre-images in  $\Phi$ , and hence the comparison is well-defined. The implementation might, however, be non-trivial. We use the following notation. For a point  $p$ ,  $(u_p, v_p)$  denotes a pre-image, and for a curve  $C$ ,  $\gamma$  denotes a pre-image, that is,  $p = \phi_S(u_p, v_p)$  and  $C(t) = \phi_S(\gamma(t))$  for all  $t \in I$ . This root concept expect the following operations:

**Compare  $u$ :** Compare the  $u$ -coordinates of two regular points  $p_1$  and  $p_2$ ; return  $\text{cmp}_u(p_1, p_2)$

**Compare  $uv$ :** Compare two regular points lexicographically, first by their  $u$ -coordinates, and in case they are equal, by their  $v$ -coordinates. This predicate is used to maintain the order of regular event points.

**Obtain minimum (resp. maximum) endpoint:** For a given curve  $C$ , return the lexicographically smaller (resp. larger) endpoint of a  $u$ -monotone curve, if the curve is closed at this end.

**Is vertical:** Determine whether a  $u$ -monotone curve is vertical.

**Compare  $v$  at  $u$ :** Given a  $u$ -monotone curve  $C$  in  $S$  and a regular point  $p$  in  $S$ , such that  $u_p$  lies in the  $u$ -range of  $\gamma$ , determine whether the pre-image of  $p$  is above, below, or lies on  $\gamma$ . More precisely, if  $C$  is vertical, determine  $v_p < v(\gamma(0))$ ,  $v_p > v(\gamma(1))$  or in between. Otherwise, since  $u(\gamma(0)) \leq u_p \leq u(\gamma(1))$  and  $\gamma$  is  $u$ -monotone, there is a unique  $t' \in [0, 1]$  with  $u(\gamma(t')) = u_p$ . Return  $\text{cmp}_v(p, \gamma(t'))$ . This predicate is used to insert a new curve with minimal end at  $p$  into the status structure.

**Compare  $v$  to right of  $u$ :** Given two  $u$ -monotone curves  $C_1$  and  $C_2$  that share a common minimal end at a point  $p$ , determine the order of the curves immediately to the right of  $p$ . The construction coming next is typical for our approach. In order to determine the ordering in which curves emanate from a point  $p$  in increasing direction of  $u$ , we compare the curves infinitesimally

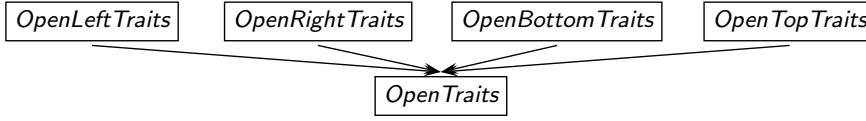


Figure 10: Bottom portion of the refinement hierarchy of the geometry-traits concept for curves embedded on an open surface, for instance, the entire plane.

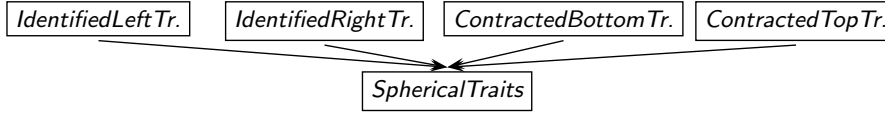


Figure 11: Bottom portion of the refinement hierarchy of the geometry-traits concept for curves embedded on a spherical-like parameterized surface.

to the right of  $p$ . More precisely, return  $\text{cmp}_v(\gamma_1(\epsilon_1), \gamma_2(\epsilon_2))$ , where  $\epsilon_1, \epsilon_2 > 0$  are infinitesimally small and  $u(\gamma_1(\epsilon_1)) = u(\gamma_2(\epsilon_2))$ . This predicate is used to insert new curves into the status structure, when the minimal end lies on an existing curve in the status structure.

**Compare  $v$  to left of  $u$ :** Symmetric to the preceding predicate. This predicate is optional and, if implemented, increases efficiency of some algorithms.

**Intersect:** Compute the intersections of two  $u$ -monotone curves  $C_1$  and  $C_2$ .

**Split:** Split a curve  $C$  at a regular point  $p$  which must lie in the interior of  $C$ . This is used when a curve is inserted that has an endpoint  $p$  in the interior of an already existing curve.

**Are mergeable:** Determine whether two curves  $C_1$  and  $C_2$  that share a common endpoint can be merged into a single continuous curve representable by the traits class.

**Merge:** Merge two mergeable curves  $C_1$  and  $C_2$  into a single curve  $C$ .

#### 4.2. The Refined Geometry-Traits Concepts

We descend to the bottom level of the hierarchy. For each side of the parameter space there are four available concepts: *OpenSideTraits*, *ContractedSideTraits*, *ClosedSideTraits*, and *IdentifiedSideTraits*.<sup>6</sup> Theoretically, this makes for a total of  $4^4 = 256$  combinations. Only a subset of them is meaningful. If a side is identified, the opposite side must also be identified. This implies that there are exactly 10 possible combinations of left and right sides, and similarly for the bottom of top sides, which makes a total of 100 combinations. A suitable geometry-traits component for unbounded curves in the plane is a model of the *OpenTraits* concept

<sup>6</sup>Note that we explicitly distinguish closed-only, contracted, and identified sides, although the latter two are closed in  $\Phi$  as well.

shown in Figure 10, while one for surfaces homeomorphic to a sphere can be a model of the combined concept *SphericalTraits* shown in Figure 11. Similarly, one can derive an own concept that suits a chosen parameterization.

The shared requirements for the four options of a side are collected in abstract layers called *HasSpecialSideTraits*, *PointOnSideTraits*, and *ComparePointOnSideTraits* shown in grey in Figure 9. The first concept collects the operations required by all side-specific refinements, and the other two provide operations that handle points, curve-ends, and curves that are contained in the boundary of the domain.

The concept *HasSpecialSideTraits* requires the following predicates:

**Locate curve-end in  $u$  (resp.  $v$ ):** Given a curve  $C$  and an index  $d \in \{0, 1\}$ , determine the location of the  $d$ -end of  $C$  with respect to the  $u$ -axis. More precisely, determine whether  $\lim_{t \rightarrow d} u(\gamma(t))$  is equal to  $u_{\min}$ , equal to  $u_{\max}$ , or falls in between. These predicates determine the location of a curve-end in parameter space.<sup>7</sup>

The next two operations determine the order of curve-ends lying on the boundary with respect to regular points and among each other. Figure 4a illustrates these comparisons.

**Compare  $u$  near boundary (resp.  $v$ ):** Given two curves  $C_1$  and  $C_2$  and an index  $d \in \{0, 1\}$  that identifies either the minimum ends or the maximum ends of the two curves, respectively, compare the  $u$ -coordinate of the curves near their limit. For an open side, a precondition assures that the  $u$ -coordinates of the curves at their limits are equal. That is, the predicate **Compare  $u$  limit on boundary** (see *OpenSideTraits* below) applied to  $C_1$ ,  $C_2$ , and  $d$  evaluates to “equal”. For a closed or identified side, again a precondition assures that the  $u$ -coordinates of the curves at their ends are equal. That is, the predicate **Compare  $u$  on boundary** (see *ComparePointOnSideTraits* below) applied to the two points, which are the  $d$ -ends of  $C_1$  and  $C_2$ , respectively, evaluates to “equal”.

The concept *OpenSideTraits* requires three additional predicates:

**Is closed:** Given a curve  $C$  and an index  $d \in \{0, 1\}$ , determine whether the pre-image of  $C$ 's  $d$ -end belongs to the domain of  $\gamma$  or not. This predicate tells whether the  $d$ -end of  $C$  is a point (closed) or open, otherwise.

**Compare  $u$  limit on boundary:** There are different predicates for  $u$  and  $v$ , due to the asymmetry mentioned in Definition 3.1. This predicate is overloaded with two versions as follows:

- (1) Given a regular point  $p$ , a curve  $C$ , and an index  $d \in \{0, 1\}$ , compare  $u_p$  and  $\lim_{t \rightarrow d} u(\gamma(t))$ . A precondition assures that  $C$  has a vertical asymptote at its  $d$ -end; that is  $\lim_{t \rightarrow d} u(\gamma(t))$  is a real value.

---

<sup>7</sup>We only give the definition with respect to the  $u$ -dimension, in case the  $v$ -version is symmetric. Otherwise, we report additional details.

- (2) Given two curves  $C_1$  and  $C_2$  and indices  $d_1, d_2 \in \{0, 1\}$ , compare the  $u$ -coordinates of the curves at their respective limits. More precisely, compare the values  $\lim_{t \rightarrow d_1} u(\gamma_1(t))$  and  $\lim_{t \rightarrow d_2} u(\gamma_2(t))$ . A precondition assures that  $C_1$  and  $C_2$  have vertical asymptote at their respective  $d$ -ends; that is  $\lim_{t \rightarrow d_1} u(\gamma_1(t))$  and  $\lim_{t \rightarrow d_2} u(\gamma_2(t))$  are real values.

**Compare  $v$  limit on boundary:** Given two curves  $C_1$  and  $C_2$ , and a single index  $d \in \{0, 1\}$  that identifies either the minimum ends or the maximum ends of the two curves, respectively, compare the  $v$ -coordinates of the curves at their respective limits. More precisely, compare the values  $\lim_{t \rightarrow d} v(\gamma_1(t))$  and  $\lim_{t \rightarrow d} v(\gamma_2(t))$ . If the two limits evaluate to infinity and have the same sign, i. e., they both evaluate to  $-\infty$  or to  $+\infty$ , then compare  $\lim_{t \rightarrow d} \frac{v(\gamma_1(t))}{v(\gamma_2(t))}$  and 1.

A closed or contracted side may contain points. The concept *PointOnSideTraits* requires two additional predicate:

**Locate point in  $u$  (resp.  $v$ ):** This predicate is similar to **Locate curve-end in  $u$** . Given a point  $p$  determine the location of its pre-image in the domain  $\Phi$  along the  $u$ -dimension. More precisely, check whether  $u_p$  is equal to  $u_{\min}$  or equal to  $u_{\max}$ , or falls in between. This function is only applied if one of the opposite vertical sides is contracted or closed. The pre-image of a contracted point is the entire side, and hence the outcome is independent of the choice of a particular pre-image, while the pre-image of a point in a closed side is unique.

A model of the concept *ComparePointOnSideTraits* must be able to order points that lie on the image of a closed or identified boundary-side; it is not needed for contracted sides, as they map to a single point on the surface. The concept requires the following additional pair of predicates:

**Compare  $u$  on boundary (resp.  $v$ ):** Given two points  $p_1$  and  $p_2$ , such that  $v_{p_1} \in \{v_{\min}, v_{\max}\}$  or  $v_{p_2} \in \{v_{\min}, v_{\max}\}$  (resp.  $u_{p_1} \in \{u_{\min}, u_{\max}\}$  and  $u_{p_2} \in \{u_{\min}, u_{\max}\}$ ), compare  $u_{p_1}$  and  $u_{p_2}$ . A point on the identification curve has two pre-images, one for  $v_{\min}$  and one for  $v_{\max}$ . The  $u$ -coordinates of both pre-images are the same and so the operation is well-defined.

We are almost done. For a contracted boundary side no additional operation (beyond those required by the concept *PointOnSideTraits*) are needed. For a closed side, we have to handle curves whose pre-image is fully contained in the boundary of the parameter space. Therefore, the concept *ClosedSideTraits* requires yet another pair of predicates:

**Locate curve in  $u$  (resp.  $v$ ):** Locate a curve  $C$  along the  $u$ -dimension. More precisely, determine whether  $u(\gamma(t))$  is equal to  $u_{\min}$ , equal to  $u_{\max}$ , or falls in between for all  $t \in (0, 1)$ .

In contrast, handling identified boundary sides demand a slightly different predicate. Thus, the concept *IdentifiedSideTraits* introduces the following requirement:

**Is on  $u$ -identification (resp.  $v$ ):** Given a point  $p$  (respectively a curve  $C$ ), determine whether  $p$  (respectively  $C$ ) lies in the image of the vertical and identified sides of the boundary. More precisely, determine whether  $u_p \in \{u_{\min}, u_{\max}\}$  for all pre-images of  $p$ . Similar for all points of  $C$ , respectively.

### 4.3. The Topology-Traits Concept

We discuss the topology-traits concept. A model of this concept must define a nested type for the EDCEL. The topology traits maintains the EDCEL and additional status information, in particular, the sorted sequence of vertices lying on curves of identification, information about the kinds (i. e., open, closed, contracted, and identified) of the sides, and the fictitious/non-fictitious distinction for vertices, halfedges, and faces. An EDCEL record (i. e., vertex, halfedge, or face) *relates* either to the interior or the boundary of the parameter space. A vertex and halfedge relates to the interior if its pre-image lies in the interior of the parameter space and to the boundary otherwise. A face relates to the interior if all bounding halfedges relate to the interior. Otherwise, it relates to the boundary. Features related to the interior are handled as in the case of a bounded subdivision of the plane. New functionality is needed for the records relating to the boundary, in particular surface-specific support to determine whether the insertion of a curve makes a component non-contractible; see Section 3.2.4. The full concept is described in Appendix A.

## 5. Concretizations

Several concretizations of the framework already exist in terms of software. The companion paper [7] discusses arrangements on spheres, quadrics, and ring Dupin cyclides. The `Arrangement_on_surface_2` package contains two topology-traits classes for the plane: one for bounded curves, which maintains a single unbounded face, and one for the unbounded curves, which uses an implicit bounding rectangle; see Section 3.2.4 and [33] for more details. Most geometry-traits classes that used to support only bounded planar curves in previous versions have been enhanced to support unbounded curves as well, for instance, the prototypical implementation for algebraic curves of any degree [6, 16]. New geometry-traits classes have been developed from scratch, for example, for geodesic arcs embedded on the sphere, and for segments, rays, and lines in the plane. With the latter, we are able to compute lower envelopes of planes in three-dimensional space [29], and, through a well-known transformation [15], Voronoi diagrams of points in the plane; see [31] for details. The diagrams are represented as planar arrangements of unbounded curves; Figure 12 shows some examples. We have chosen degenerate input sets to highlight that our framework handles arbitrary degeneracies.

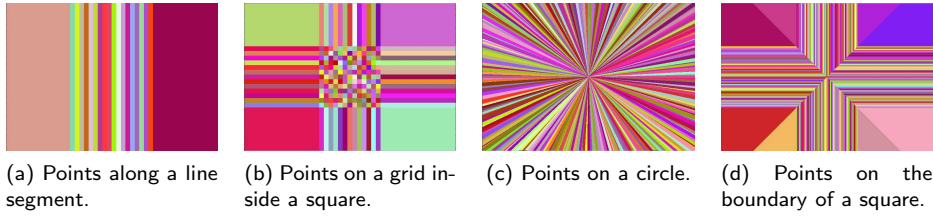


Figure 12: Voronoi diagrams of points in the unbounded plane.

## 6. Conclusions and Future Work

We describe a general framework for the construction, maintenance, and manipulation of arrangements induced by curves embedded on a class of two-dimensional orientable parametric surfaces. We reuse existing code for arrangements of bounded curves in the plane, generalize it, and complete the implementation by specific code (encapsulated in traits classes) that handles the particular surface and curves. Code reuse minimizes the effort required to develop traits classes for new families of curves and new surfaces. Such developments benefit from a highly generic and efficient code base for the main arrangement-related algorithms, which supports a broad set of features. Clearly, developing surface-specific traits-classes is a relatively small task compared to developing a full implementation from scratch. We use advanced techniques from the generic and the object-oriented programming paradigms [32] to achieve maximum efficiency, flexibility, and robustness.

**Future work:** With the topology concept, we successfully separated topological operations for maintaining a surface-specific EDCEL from surface-independent ones. However, topology-traits classes for different surfaces (see also [7]) show similarities. For example, in case of an identification curve, they all maintain a sorted sequence of points. Likewise, the decision with respect to face splits and their CCBs rely on similar information. We therefore believe that a further unification should be possible. For example, we envision a model that can be configured for various topologies by just naming what happens on the boundaries of the parameter space.

In this work, we also restricted ourselves to the single-domain case, that is  $\Phi = U \times V$ . Another future goal is to extend the framework to handle general orientable surfaces, which can be conveniently represented by a collection of domains, each of which is supported by a rectangular parameter space. It is known which polygonal maps give rise to orientable surfaces and each orientable surface has a normal form, which already includes surfaces of higher genus [26]. In addition, one may wish to consider surfaces with singularities (e. g., the pinch point of a double cone), which require to decompose them, such that singularities only appear on the boundary of the parameter spaces. Concerning the framework, the different

individually obtained parameter spaces are glued together according to the topology of the surface, and therefore will naturally be described in, and handled by, an extension of the topological concept. A key step is to derive additional geometric predicates for such *stitched boundaries*. Also, the EDCEL needs to be replaced by a more powerful data structure.

Arrangements on surfaces can also be a tool in other settings. Consider, for example, the task of computing *three-dimensional arrangements of surfaces*. As a first step, one could compute, for each surface, the arrangement defined by its intersection curves with the other surfaces. This step is supported by our package. As a further step, one needs to identify equal vertices and edges on different surfaces. How to do this for quadrics has been shown in [24]. His approach uses a direct parameterization of the quadrics. However, the important subtask, namely the equality-detection of vertices and edges can be formulated almost abstractly. We want to explore whether this identification can be done for the surfaces on which we can compute arrangements. It might be required to add additional geometric primitives that determine the equality of two geometric objects on two different surfaces.

## Acknowledgments

We thank Michael Hemmer for a fruitful discussion on the design of the concepts and Michael Kerber and Ophir Setter for commenting on draft versions of the paper.

## References

- [1] CGAL *User and Reference Manual*, 3.4 edition, 2008. [http://www.cgal.org/Manual/3.4/doc\\_html/cgal\\_manual/packages.html](http://www.cgal.org/Manual/3.4/doc_html/cgal_manual/packages.html).
- [2] Pankaj K. Agarwal and Micha Sharir. Arrangements and their applications. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, chapter 2, pages 49–119. Elsevier Science Publishers, B.V. North-Holland, Amsterdam, North-Holland, 2000.
- [3] Marcus V. A. Andrade and Jorge Stolfi. Exact algorithms for circles on the sphere. *International Journal of Computational Geometry and Applications*, 11(3):267–290, June 2001.
- [4] Matthew H. Austern. *Generic Programming and the STL*. Addison-Wesley, 1999.
- [5] Jon L. Bentley and Thomas Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, 28(9):643–647, 1979.
- [6] Eric Berberich and Pavel Emeliyanenko. CGAL’s Curved Kernel via Analysis. Technical Report ACS-TR-123203-04, Algorithms for Complex Shapes, 2008.
- [7] Eric Berberich, Efi Fogel, Dan Halperin, Michael Kerber, and Ophir Setter. Arrangements on parametric surfaces II: Concretization and applications, 2010. Accepted to *Mathematics in Computer Science*.

- [8] Eric Berberich, Efi Fogel, Dan Halperin, Kurt Mehlhorn, and Ron Wein. Sweeping and maintaining two-dimensional arrangements on surfaces: A first step. In *Proceedings of 15th Annual European Symposium on Algorithms (ESA)*, volume 4698 of *LNCS*, pages 645–656. Springer-Verlag, 2007.
- [9] Eric Berberich, Michael Hemmer, Lutz Kettner, Elmar Schömer, and Nicola Wolpert. An exact, complete and efficient implementation for computing planar maps of quadric intersection curves. In *Proceedings of 21st Annual ACM Symposium on Computational Geometry (SoCG)*, pages 99–106. Association for Computing Machinery (ACM) Press, 2005.
- [10] Eric Berberich and Michael Kerber. Exact arrangements on tori and Dupin cyclides. In *Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling (SPM)*, pages 59–66. Association for Computing Machinery (ACM) Press, 2008.
- [11] Erik Brisson. Representing geometric structures in  $d$  dimensions: topology and order. In *SCG '89: Proceedings of the fifth annual symposium on Computational geometry*, pages 218–227, New York, NY, USA, 1989. ACM.
- [12] Frédéric Cazals and Sébastien Lorient. Computing the arrangement of circles on a sphere, with applications in structural biology. *Computational Geometry: Theory and Applications*, 42(6-7):551–565, 2009.
- [13] Mark de Berg, Mark van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition, 2000.
- [14] Pedro M.M. de Castro, Frédéric Cazals, Sébastien Lorient, and Monique Teillaud. Design of the CGAL 3D Spherical Kernel and application to arrangements of circles on a sphere. *Computational Geometry: Theory and Applications*, 42(6-7):536–550, 2009.
- [15] Herbert Edelsbrunner and Raimund Seidel. Voronoi diagrams and arrangements. *Discrete & Computational Geometry*, 1:25–44, 1986.
- [16] Arno Eigenwillig and Michael Kerber. Exact and efficient 2D-arrangements of arbitrary algebraic curves. In *Proceedings of 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 122–131, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics (SIAM).
- [17] Andreas Fabri, Geert-Jan Giezeman, Lutz Kettner, Stefan Schirra, and Sven Schönherr. On the design of CGAL a computational geometry algorithms library. *Software — Practice and Experience*, 30(11):1167–1202, 2000.
- [18] Efi Fogel, Dan Halperin, Lutz Kettner, Monique Teillaud, Ron Wein, and Nicola Wolpert. Arrangements. In J.-D. Boissonat and M. Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, chapter 1, pages 1–66. Springer-Verlag, 2007.
- [19] Efi Fogel, Ophir Setter, and Dan Halperin. Exact implementation of arrangements of geodesic arcs on the sphere with applications. In *Abstracts of 24th European Workshop on Computational Geometry*, pages 83–86, 2008.
- [20] Efi Fogel, Ophir Setter, and Dan Halperin. Movie: Arrangements of geodesic arcs on the sphere. In *Proceedings of 24th Annual ACM Symposium on Computational Geometry (SoCG)*, pages 218–219. Association for Computing Machinery (ACM) Press, 2008.

- [21] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns — Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1999.
- [22] Dan Halperin. Arrangements. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 24, pages 529–562. Chapman & Hall/CRC, 2nd edition, 2004.
- [23] Dan Halperin and Christian R. Shelton. A perturbation scheme for spherical arrangements with application to molecular modeling. *Computational Geometry: Theory and Applications*, 10:273–287, 1998.
- [24] Michael Hemmer. *Exact Computation of the Adjacency Graph of an Arrangement of Quadrics*. Ph.D. thesis, Johannes-Gutenberg-Universität, Mainz, Germany, 2008.
- [25] Younis O. Hijazi and Thomas M. Breuel. Computing arrangements using subdivision and interval arithmetic. In *Proceedings of 6th International Conference on Curves and Surfaces*, pages 173–182, 2006.
- [26] Francis Lazarus, Michel Pocchiola, Gert Vegter, and Anne Verroust. Computing a canonical polygonal schema of an orientable triangulated surface. In *Proceedings of 17th Annual ACM Symposium on Computational Geometry (SoCG)*, pages 80–89, 2001.
- [27] Kurt Mehlhorn and Stefan Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, UK, 2000.
- [28] Kurt Mehlhorn and Michael Seel. Infimaximal frames: A technique for making lines look like segments. *International Journal of Computational Geometry and Applications*, 13(3):241–255, 2003.
- [29] Michal Meyerovitch. Robust, generic and efficient construction of envelopes of surfaces in three-dimensional space. In *Proceedings of 14th Annual European Symposium on Algorithms (ESA)*, volume 4168 of *LNCS*, pages 792–803. Springer-Verlag, 2006.
- [30] Victor Milenkovic and Elisha Sacks. An approximate arrangement algorithm for semi-algebraic curves. *International Journal of Computational Geometry and Applications*, 17(2):175–198, 2007.
- [31] Ophir Setter, Micha Sharir, and Dan Halperin. Constructing two-dimensional Voronoi diagrams via divide-and-conquer of envelopes in space. In *Proceedings of 6th Annual International Symposium on Voronoi Diagrams in Science and Engineering (ISVD)*, pages 43–52, 2009.
- [32] Ron Wein, Efi Fogel, Baruch Zukerman, and Dan Halperin. Advanced programming techniques applied to CGAL's arrangement package. *Computational Geometry: Theory and Applications*, 38(1–2):37–63, 2007. Special issue on CGAL.
- [33] Ron Wein, Efi Fogel, Baruch Zukerman, and Dan Halperin. 2D arrangements. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.4 edition, 2008.

## Appendix A. The Topology-Traits Concept

**Initialize:** Construct an EDCEL that represents an empty arrangement.

**Return fictitiousness of vertices, halfedges, and faces:** Recall that the `EDCEL` may contain records that have no geometric meaning. These functions return whether a record has geometric meaning or not. It is needed to filter out fictitious elements from traversals.

**Place a curve-end:** Given a curve-end whose pre-image lies on or emanates from the boundary and a face that contains the interior of the curve, determine the `EDCEL` vertex or (fictitious) edge that contains the given curve-end, if it exists. Otherwise, return `NULL`. The topology-traits class uses this method and the next to implement open, closed, contracted, or identified boundaries.

**Notify on boundary vertex creation:** This is called by the `Arrangement_on_surface_2` instance to notify its topology-traits instance about the creation of a new `EDCEL` vertex whose pre-image(s) belong to  $\partial\Phi$ . It enables the topology-traits class to keep its internal objects up-to-date, for instance, the ordered sequence of `EDCEL` records representing points on an identification curve, while the arrangement class can still notify its observers about the respective structural changes.

**Locate a curve-end:** Locate the `EDCEL` record that contains a given curve-end. The pre-image of the curve-end lies on the boundary or emanates from the boundary. The method returns a vertex, an edge, or a face and is used by the point location operation.

**Locate the halfedge around a boundary vertex:** For an `EDCEL` vertex whose pre-images belong to  $\partial\Phi$  return the predecessor halfedge that identifies where to insert a given curve approaching this vertex. It may return `NULL`, if there is no such halfedge.

**Split fictitious edge:** Splits a given fictitious edge into two and returns a handle to one of them. It is the topology-traits class that implements this function, as this is a structural change that relates to the boundary of the parameter space. The method is used when a curve emanating from the boundary is added to the arrangement. The next two functions are used when such a curve is deleted.

**Is redundant:** Determines whether a given vertex whose pre-image is on the boundary is redundant. A vertex becomes redundant after all non-fictitious incident edges are removed.

**Remove redundant vertex:** Remove a redundant vertex from the internal structures. Subsequent to its call, the arrangement notifies its attached observers about the removal of the vertex.

The next set of functions deals with faces:

**Is unbounded:** Determine whether a face is unbounded. This test is invoked when an unbounded face is split by a bounded curve into two to decide which of the resulting faces is still unbounded.

**Is in face:** Determine whether a given point belongs to the interior of a face, while ignoring any of its inner components, that is, whether the point is contained in the region to the left of the face's outer CCBs. It is used for

point location in general, and relocating inner CCBs after a face has split in particular.

Maintaining CCBs as explained in Section 3.2.4 can be reduced to determine whether one or two cycles of curves are contractible or not — and how often a non-contractible cycle really crosses an identification in *positive* direction: That is, when walking along a cycle counting how often the curves' pre-images in parameter space switch from left of the right side to right of the left side for identified vertical sides; and from below the top side to above the bottom side for identified horizontal sides, respectively. This is interfaced by the following operation:

**Sign of crossing:** Given two EDCEL half-edges (one can be replaced by an actual curve) each having an end on the same identification at the same point. Determine whether the crossing in the direction of the half-edge(s) at the point is positive, negative (i. e., in the other direction) or zero which indicates that both curves' pre-images emanate to the same side of the parameter space.

In addition to these functions, a model of the topology-traits concept also has to define a set of *visitors*:

**Construction sweep-line visitor:** A sweep-line visitor class for constructing the arrangement of a set of curves from scratch.

**Insertion sweep-line visitor:** A sweep-line visitor class for inserting a set of curves into a non-empty arrangement.<sup>8</sup>

**Overlay sweep-line visitor:** A sweep-line visitor class for constructing a new arrangement corresponding to the overlay of two input arrangements.

**Insertion zone visitor:** Used to insert a single curve into an existing arrangement by traversing its zone.

**Default point location:** This point-location strategy is used as fall-back if no other is selected when invoking the insertion of a curve using the zone traversal. Recall that the initial step of the zone traversal is to locate the EDCEL-record in a given arrangement containing the minimal end of the given curve.

Since the visitor classes are defined by the topology-traits classes, it is possible to support generic functions that operate on the `Arrangement_on_surface_2` class. For example, the function `insert_curves(arr, begin, end)` accepts an arrangement instance `arr` and a range of curves defined by `[begin, end)`. If the arrangement is empty, it uses the construction sweep-line visitor to sweep over the input curves and construct their arrangement. Otherwise, it uses the insertion sweep-line visitor to insert them into the existing arrangement.

---

<sup>8</sup>The concept also demands for two visitors (construction and insertion) that work on more restricted input sets. For such the curves are already known to be interior disjoint. This reduces the number of expected geometric operations, i. e., `Intersect` is obviously not required. However, the topology-traits concept makes no assumption on whether specialized implementations are provided, or one reuses (with `typedefs`) the more general visitors (knowing that, e. g., the intersection operation is never invoked).

Any topology-traits class is allowed to provide more (surface-specific) functionality — beyond expectations stated by the concept. An example is to provide methods that enable access to all vertices lying on an identification curve.

Eric Berberich

School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel  
e-mail: [ericb@post.tau.ac.il](mailto:ericb@post.tau.ac.il)

Efi Fogel

School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel  
e-mail: [efif@post.tau.ac.il](mailto:efif@post.tau.ac.il)

Dan Halperin

School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel  
e-mail: [danha@post.tau.ac.il](mailto:danha@post.tau.ac.il)

Kurt Mehlhorn

Max-Planck-Institut für Informatik, Saarbrücken, Germany  
e-mail: [mehlhorn@mpi-inf.mpg.de](mailto:mehlhorn@mpi-inf.mpg.de)

Ron Wein

School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel  
e-mail: [wein@post.tau.ac.il](mailto:wein@post.tau.ac.il)