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 interface for our framework; only the operations in the interface need to be implemented for a specific application. The companion paper [6] 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 (2000). Primary 68U05; Secondary 14Q10.

Keywords. Computational Geometry, arrangement of curves, parametric surface, CGAL, robust geometric computing.

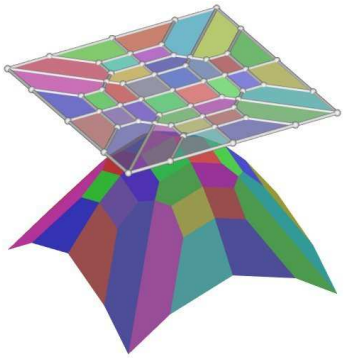
1. Introduction

We are given a surface S in $\mathbb{R}^3$ and a set $\mathcal{C}$ of curves embedded on this surface. The curves divide S into a finite number of 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 embedded on two-dimensional orientable parametric surfaces such

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 [7] was presented at ESA 2007.

as planes, cylinders, spheres, tori, and surfaces homeomorphic to them. Such arrangements have many theoretical and practical applications [1, 6, 16, 20]. 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` [31] as of version 3.4.[1] CGAL, the Computational Geometry Algorithms Library,[2] is a generic and robust, yet efficient, implementation of geometric data-structures and algorithms [15]. Our package does not only compute arrangements; it also provides useful further operations, such as point location, insertion and removal of curves, overlay computation, iteration over all features (vertices, edges, faces) and the features incident to a particular feature. 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 interface; only the operations in the interface need to be implemented for a specific application. 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 6 and the companion paper [6]. Examples are arrangement of lines in the plane, of arcs of great circles on the sphere [17, 18], of intersection curves between quadric surfaces and a fixed quadric [7], and of intersection curves between arbitrary algebraic surfaces and a fixed Dupin cyclide [9]. 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 [30]. 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 [28]. Such an arrangement has, in general, several unbounded faces. However, an arrangement of bounded curves has only a single unbounded face. Of course, preprocessing (enclosure in a bounding rectangle) and postprocessing allows one to recover the unbounded faces as shown in the figure to the right. 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.



[1] The manual of the `Arrangement_on_surface_2` package in CGAL version 3.4 does not include material for non-planar surfaces; thus, it can be used only privately. We expect that a nearby future release will support all surfaces publicly.

[2] `http://www.cgal.org`

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 [3]. 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 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 instantiate template parameters. We also make use of *design patterns* [19]. 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 [30] for more details.

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 [16]. Mehlhorn and Seel [27] 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 [2] develop exact algorithms for manipulating circular arcs on a sphere. Halperin and Shelton [22] incrementally construct arrangements of circles on a sphere. Berberich *et al.* [8] 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 Loriot [11] 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 [24] compute arrangements induced by implicit curves using a subdivision method and interval arithmetic, and Milenkovic and Sacks [29] compute arrangements using approximations. In contrast, Eigenwillig and Kerber [14] describe an exact and complete approach arrangements of algebraic curves. Their most recent implementation uses our framework.

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 a geometry and a topology concept. We give the details for both concepts in Sections 4 and 5, respectively.

Section 6 surveys already existing concretizations. We finally give some concluding remarks and future-work directions in Section 7.

2. The Basic Arrangement Framework

CGAL's `Arrangement_2` framework separates between the topological and the geometric aspects of the subdivision, that is, it separates the combinatorial, graph-like structure (the topology) from the actual embedding on the surface (the geometry). 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 with appropriate models for the parameters. The geometry-traits class introduces the C++ type names of the basic geometric objects (point, curve, 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 Section 5 for details. In particular, it maintains a representation of the arrangement graph in form of an *extended doubly-connected edge list* (EDCEL) data-structure as is suitable for the particular topology. An EDCEL is a DCEL [12, Section 2.2] with additional features for a topologically consistent representation.

2.2. The Bentley-Ottmann Sweep

Bentley and Ottmann [4] 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, new curves are added to, and swept curves are removed from the status structure, curves swap their position in the status structure, newly adjacent curves are checked for intersections to the right of the sweep line (= intersections having lexicographically larger coordinates), and 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 [26].

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 [19]. Examples of visitors [30] 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., [20]). Users may introduce their own sweep-based algorithms by implementing an appropriate visitor class.

The actual sweep is preceded by a preprocessing phase that subdivides input curves into x-monotone subcurves and isolated points and ensures further conditions that might be required by 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 adding a set of input curves to an arrangement. A different approach is to add the input curves one by one to a growing arrangement traversing the *zone* of the new curve in the arrangement. The zone [20] of an x-monotone curve C in an arrangement is the set of cells intersected by it. The zone of a curve C is computed by locating the left endpoint of C in the arrangement, 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., [12, 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.

3. Sweeping and Zoning Over Surfaces

We generalize the framework from the plane to parametric surfaces such as halfplanes, cylinders, tori, etc., and surfaces homeomorphic to these (see Figure 1). 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.

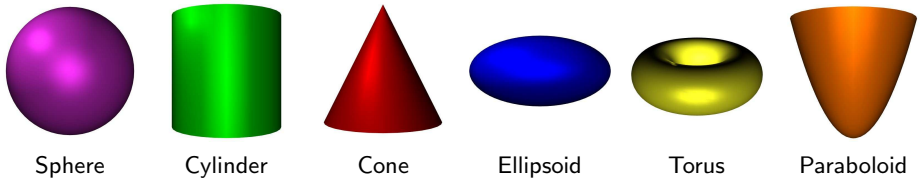


FIGURE 1. Various two-dimensional parametric-surfaces.

3.1. Parametric Surfaces and Curves Embedded into Them

We use $\mathbb{R}^*$ 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 $\mathbb{R}^*$. So the reader may also think of finite intervals instead of the (compactified) real line in what follows.

Definition 3.1 (Parametric Surface). A parametric surface S is given by a continuous function $\phi_S : \Phi \to \mathbb{R}^3$, 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 $\mathbb{R}^*$. 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 Φ 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 Φ, in particular, ϕ_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 ϕ_S being the identity mapping. For example, $f_S(u,v) = (u,v,0)$ with $U = V = (-\infty, +\infty)$ parameterizes the plane. Surfaces such as paraboloids can be modeled through continuous and bijective parameterizations, for example, $f_S(u,v) = (u,v,u^2+v^2)$, 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 (π, 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 ϕ_S is constant on it. The image of the side is called a *contraction point* of S.

Definition 3.3 (Identification). The bottom and top side of the domain (similarly for left and right side) 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 the horizontal sides and, in addition, contracting one of the pairs. 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 data structure would be needed, such as a cell-tuple structure [10]. This cannot be handled by our framework.

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 emanate from an open side of Φ. Curves must have only a finite number of self-intersections.

Definition 3.4 (Curve). A *parameterizable curve* γ is a continuous function $\gamma : I \to \Phi$, where I is an open, half-open, or closed interval with endpoints 0 and 1, and γ is injective except for a finite number of points. If $0 \notin I$, $\lim_{t \to 0+} \gamma(t)$ exists (in the closure of Φ) and lies in an open side of the boundary. Similarly, if $1 \notin I$, $\lim_{t \to 1-} \gamma(t)$ exists and lies in an open side of the boundary. A curve C in S is the image of a curve γ 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 γ 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 Φ. 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 γ. 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 γ is the point $(-\pi, 0)$ in Φ and a point on the equator of the sphere. It is closed on the sphere, but non-closed in Φ. 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 γ is incident to the left side if either $d \in I$ and $\gamma(d)$ lies on the left side or $d \notin I$ and $\lim_{t\to d} \gamma(t)$ lies on the left side, which is then an open side. (Similarly for the other sides).

We can now formally state the goal of our work: compute the arrangement defined by a set of curves on a surface. The surface must be parameterizable as defined above and the curves must be decomposable into parameterizable curves. Any two curves in the set intersect only a finite number of times and overlap only in a finite number of sections.[3] We also need that our curves are *nice* in the sense that all geometric operations defined in Section 4 can be defined for them. We do not define this notion formally, but only state that, for instance, algebraic curves are nice, while most curves based on trigonometric functions are not. Since we require subcurves to be u-monotone and sweepable curves, then such function graphs (for trigonometric functions for example) must be decomposed into an infinite number of subcurves, for which we need the following notion:

Definition 3.5 (u-monotone curve). A *strongly u-monotone curve* is the image of a curve γ, such that if $t_1 < t_2$, then $u(\gamma(t_1)) < u(\gamma(t_2))$ for $t_1 < t_2$. A *vertical curve* is the image of a curve γ, such that $u(\gamma(t)) = c$ for all $t \in I$ and some $c \in U$ 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. 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 $t \in (0, 1)$.

In a preprocessing step all input curves are decomposed into sweepable subcurves; see also Section 3.2.2. Sweepable curves are parameterizable.

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 curves embedded on it?* We (conceptually) sweep the parameter space Φ with a vertical line ℓ from $u_{\min}$ to $u_{\max}$. The sweep of Φ induces a sweep of the surface through the parameterization ϕ_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

[3] In this paper, we do not discuss overlap between curves. The implementation handles them.

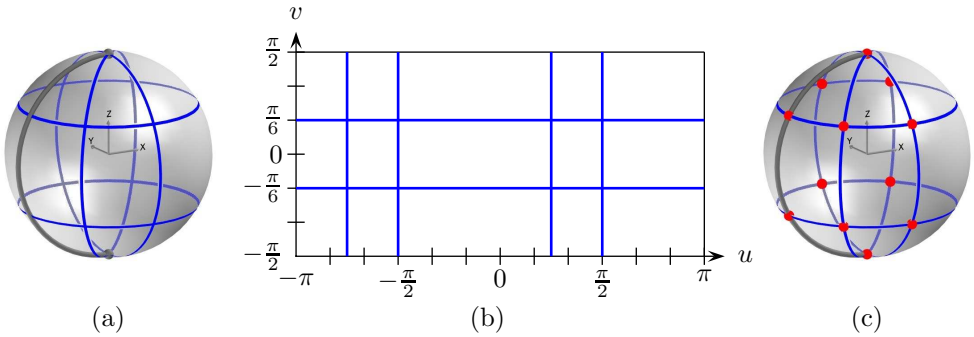


FIGURE 2. (a) A sphere with two great circles passing through the poles and two circles parallel to the equator. The identification curve is shown in grey. (b) The curves' pre-images in parameter space. (c) The EDCEL. Observe that we have two vertices on the identification curve.

in lexicographic increasing order. However, we also need to take care of additional issues; see Figure 2 for an illustration of the second and third item.

(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. We will make our definitions such that non-regular points can mimic their multiple pre-images.

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

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 emanates from a side of the domain. An intersection point in the interior of a curve is necessarily regular, as we sweep only sweepable curves. We will 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 Φ, we sweep over its interior $\Phi \setminus \partial\Phi$, or alternatively viewed,

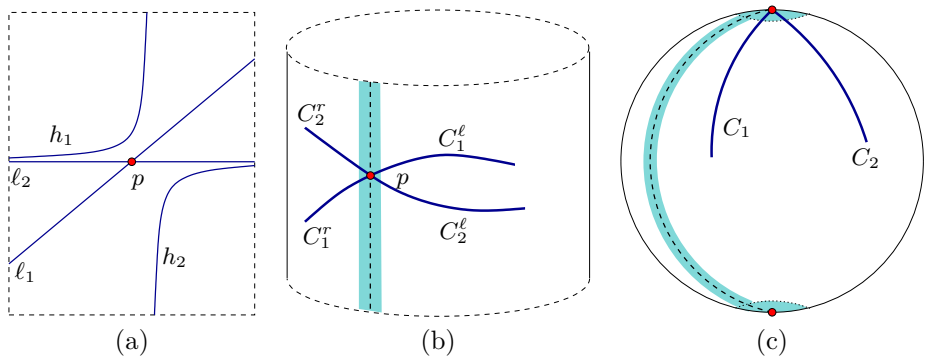


FIGURE 3. Comparing sweep events. (a) The order of the events is: minimal end of ℓ_1, minimal end of ℓ_2, minimal end of h_1 (all left side of boundary), maximal end of h_1 (top side of boundary), intersection of ℓ_1 and ℓ_2 at p (interior), minimal end of h_2 (bottom side of boundary), maximal end of h_2, maximal end of ℓ_2, and maximal end of ℓ_1 (all right side of boundary). (b) Comparing near the identification curve: $C_2^\ell < C_1^\ell$ right of red point. (c) Comparing near a point of contraction: maximal end of vertical C_1 is smaller than maximal end of vertical C_2

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 Φ. 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 3(b), the curves C_1 and C_2 cross the curve of identification. Their preimages in parameter space are curves γ_1 and γ_2 having their 0-end on the left side and their 1-end on the right side. A curve in S that winds around the cylinder several times will give 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 event. 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 this 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 a curve-end that emanates from the left side is smaller than any point lying in the interior of the parameter space which in turn is smaller than any curve-end going to the right side. We compare two curve-ends emanating from the left side by considering their intersections with a vertical line $u = u_0$ for small enough u_0

and return the v-order of these points; "small enough" means that the result does not depend on the choice of u_0. Similar rules apply to the other situations; see Figure 3 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 processing the first event in the queue and proceeding one at a time, we consider the initial segment of the event queue that consists of curve-ends emanating from the left side. We insert them into the status structure, copying the order from the event queue, avoiding further geometric comparisons. Similarly, towards the end of the process, when the event queue contains only curve-ends ending in the right side, we remove all these events in one blow from the event queue.

3.2.4. Constructing and Maintaining the EDCEL**.** 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 ordered in a natural way, namely 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 half-edge (q, r) where (q, r) is the edge following (q, p) in the counterclockwise 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 a path. Isolated vertices form trivial components. Any non-trivial component decomposes into cycles of halfedges induced by the successor relation. The halfedges in each such cycle have the same face to their left. This face is stored with each halfedge. The boundary of each face consists of a number of such cycles; we call them the CCBs (*connected components of the boundary*) of the face. A face 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 its parameter space, whether it is open, closed, contracted, or identified. It maintains the representation of the arrangement as an EDCEL. It also maintains fictitious nodes and edges representing open boundaries (see below), vertices for contraction points and, for each identification curve, the sorted sequence of DCEL records representing points on this curve. All of this (and some more on nesting; see below) constitutes the prefix E in EDCEL. The algorithms and the topology-traits class communicate with each other through methods provided by the latter. For example, when two sweep-events correspond to the same point in S, the sweep algorithm is not aware of this effect and the topology-traits class deals with it.

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

Contractions and Identifications: Consider a sweep of the sphere with some circles embedded into it; see Figure 2. In parameter space, a great circle passing through

the poles is a pair of vertical segment having their curve-ends on the lower and upper side and a circle parallel to the equator is a horizontal line having its curve-ends on the left and right side. 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 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. The 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 node at a certain v-value is created as usual and also recorded in the sorted sequence. When this node is to be created for the second time because its second pre-image is encountered by the sweep, the topology-traits class notices that the node already exists and hence do not create a new node. Rather, the already existing node is used instead. In this way, the proper identifications are made.

Open Boundaries: The DCEL data-structure is designed for representing bounded arrangements in the plane and related topological structures. It was not designed to deal with open curve-ends. We have to treat open curve-ends, for instance, Voronoi diagrams have rays going off to infinity. We want to handle open curve-ends with as little additional code as possible, for example, the traversal of the boundary of a face in a Voronoi diagram 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 rectangle (or cycle) at infinity. More consistent solutions exist and we have experimented with more. Both solutions have a simple geometric interpretation.

In the first solution (*single vertex at infinity*) we view the plane as the image of a punctured sphere (= sphere with the north pole removed) under stereographic projection. The north pole itself gives rise to a single vertex at infinity, say V_{inf}. All unbounded curves are incident to it and the cyclic ordering of these curves around V_{inf} is well-defined. So in our eDCEL, any unbounded face would have an extra vertex, namely V_{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_{inf} being fictitious.

In the second solution (*implicit bounding rectangle*) we essentially view 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 fictitious all vertices and edges. In this representation, there is also a fictitious face, the "outside" of the fictitious 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 discussion above readily extends to other surfaces. For example, in the case of a cylinder, one would have two fictitious circles. In Figure 3(b), we would have a fictitious circle for the upper and lower rim of the cylinder, respectively. In the case of the paraboloid $z = x^2 + y^2$, we would have one circle at infinity.

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 runs along the boundary. In fact, such an additional curve would foreclose the possibility to have a face incident to the boundary which does not contain a curve along the image of the domain's boundary.

Face Types, Nesting, Inner and Outer CCBs: The faces in an arrangement will 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 and will also classify the CCBs incident to a face into outer or inner CCBs. Nesting is exploited in a number of algorithms, for example point location and face traversal. We will see below how the knowledge of face types and nesting simplifies the update step after the addition of a curve to the arrangement.

Recall that the EDCEL-graph decomposes into connected components. 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. A face and a component are adjacent in this graph if they are incident to each other in the arrangement. The face-component graph is connected. Below, we will classify the CCBs of any face as *outer* and *inner*. This information can be encoded into the face-component graph by directing the edges accordingly. Assume F and K are incident. 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.

For a component K, we call the maximal connected subsets of $S \setminus K$ *regions*. Any face of our arrangement is *contained* in one of these regions. For any face F incident to K, one or more CCBs of F are *contained* in K.

Let us start with the familiar case of bounded curves in the plane and let K be any component. According to Jordan's curve theorem, one region 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; see Figure 4. We extend nestedness to faces. For each i, there is a unique face F_i in our arrangement that is incident to K and contained in R_i. Then, the F_i, $i \geq 1$, are nested in F_0. Let B_i be the CCB of F_i contained

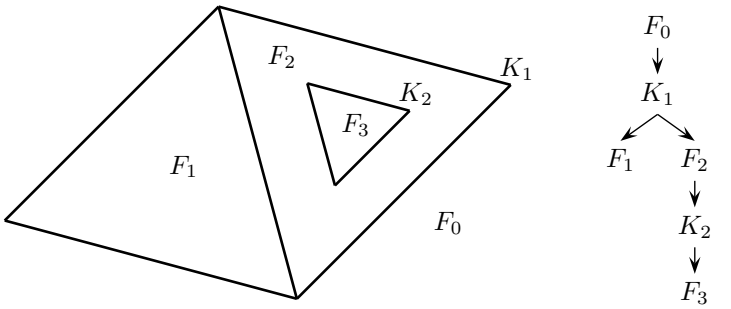


FIGURE 4. A planar embedded graph and the corresponding directed face-component incidence graph.

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 and the arrangement has a unique unbounded face. The unbounded face is the root of the directed face-component graph.

On a sphere, there is no natural distinction between the regions incident to a component. However, there is a well-known remedy. We fix a reference point on the sphere (not lying on any edge or vertex), and call the region containing the reference point "unbounded". In this way, we have a well-defined notion of nesting. The reference point is commonly refered to as the north pole. Since we use the north pole as a point of contraction, we use a different point as our reference point. We designate the face containing the area near $(u_{\max}, v_{\max})$ as the special face. There is another natural 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.

Let us next consider an unbounded component in the plane. There are natural ways of defining a nesting, corresponding to our ways of handling edges going to infinity. Having a single point of infinity corresponds to viewing the plane as the stereographic projection of a sphere. So we are in the situation discussed in the preceding paragraph. One of the unbounded faces is special and all other faces are nested in it. Having a circle at infinity corresponds to viewing the plane as the projection of the lower hemisphere; the circle at infinity is the projection of the equator and 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. The same strategies are applicable to bounded surfaces homeomorphic to an (open) disc.

We turn to the torus; see Figure 5. 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. On the torus, being a surface of genus one, there are two essentially different types of non-contractible curves. The identification curves are examples for them. For a closed curve C it is

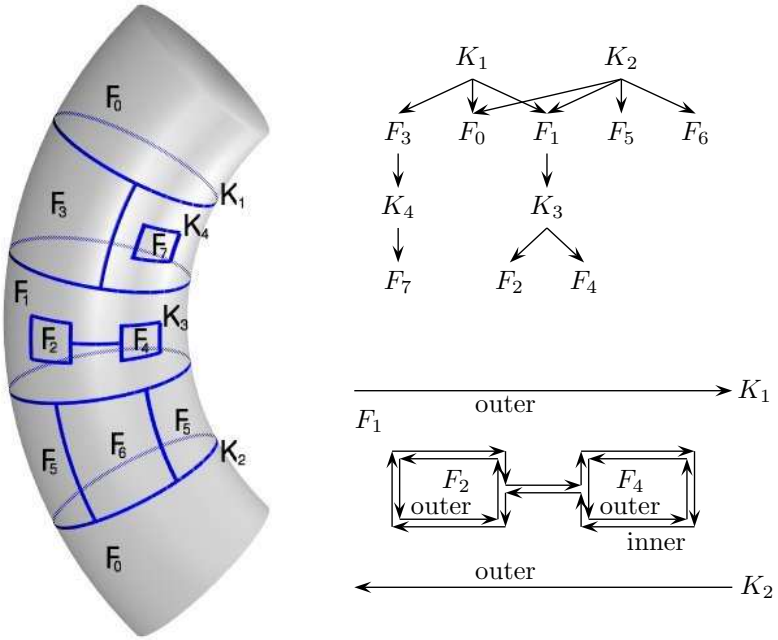


FIGURE 5. A graph embedded on a (partially displayed) torus and the corresponding directed face-component incidence graph. F_0 and F_1 are cylinder-like and all other faces are disk-like. F_0 and F_1 are have two outer CCBs each, one contained in K_1 and one contained in K_2; F_1 also has one inner CCB. The lower drawing on the right shows a close-up view of the CCBs of F_1, F_2 and F_4.

easy to decide whether it is contractible. We simply count its number of intersections with the curves of identification. Then, C is non-contractible if and only if this number is odd.

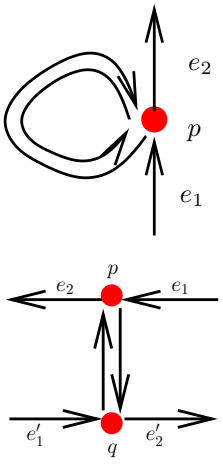
We call a component *contractible*, if no non-contractible closed curve is contained it it. Consider now the torus and the components of the arrangement graph. If all components are contractible, all faces of the arrangement except for one are punctured disks and one face, say F_0, is a punctured torus. 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 $K_1, K_2, \ldots, K_\ell$. If some component contains both kinds of non-contractible closed curves, $\ell = 1$ and all regions of $S \setminus K_1$ are disk-like. The CCBs contained in K_1 are outer CCBs for these faces. Within each such face, we have the situation of the plane with bounded components. So assume that no component contains both kinds of non-contractible cycles. Then, all non-contractible

components contain the same kind of non-contractible cycle, as instances of different kinds necessarily cross. $S \setminus (K_1 \cup \ldots \cup K_\ell)$ consists of disk-like and cylinder-like regions. Disk-like regions are incident to exactly one of the K_i's and cylinder-like regions are incident to two. For the faces corresponding to disk-like regions, the unique CCB contained in one of the K_i's is called outer. For the faces incident to two K_i's, we call the CCBs contained in both K_i's outer. The face-component graph consists of a cycle containing the non-contractible K_i's and the cylinder-like faces and trees attached to these components and faces. In the directed component-face graph, the K_i's form the top level, the cylinder-like faces have two parents, and the disk-like faces have one.

The treatment of the cylinder is a special case of the discussion of the torus. In the face-component graph, the cylinder-like faces and the components separating them form a chain instead of a cycle as in the case of the torus. We mention that one might also view the sphere as being punctured at the poles and then treat it like a cylinder.

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 half-edges (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 half-edges. 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 will insert (p, q) followed by (q, p) into one of the existing CCBs.



It remains to discuss the case that both p and q have degree at least two after the addition, $p = q$ is possible. We first show how to update the CCBs and then how to update the face-component graph.

Consider the situation at p and q. If $p = q$, and p was an isolated node before the addition, we create two new CCBs, both consisting of a single half-edge (p, p). They have opposite directions. If $p = q$, and p was not an isolated node before the addition, two cases can occur. Either an existing CCB is split into two (this is as the case $p \neq q$ discussed below) or one copy of (p, p) is inserted into an existing CCB and the other one forms a CCB of its own. In the figure on the left, one copy of (p, p) is inserted between e_1 and e_2 into an existing CCB and the other copy of (p, p) forms a CCB of its own. In general, we will have $p \neq q$ and there will be edges incident to p and q before the addition. Assume that (p, q) becomes the successor of half-edge e_1 and (q, p) becomes the successor of e'_1. The old successor, say e_2, of e_1 becomes the successor

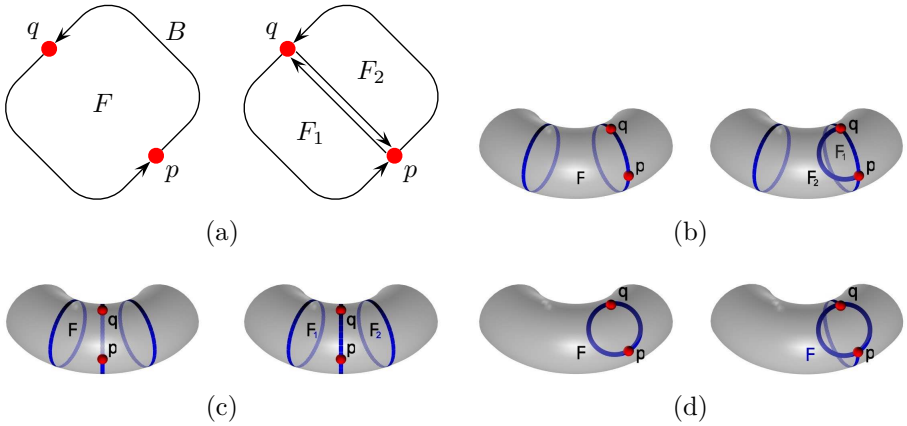


FIGURE 6. The number of CCBs increases by the addition of $\{p, q\}$. (a) and (b) illustrate the case where the type of the component containing p and q does not change. In (b), a cylinder-like face F is split into a cylinder-like F_2 and a disk-like face F_1. In (c), a cylinder-like face is split into two cylinder-like faces, and in (d), a torus-like face is converted into a cylinder-like face.

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 $e_1 \to (p, q) \to e'_2$ and $e'_1 \to (q, p) \to e_2$ 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 either the case when $p = q$ and p was an isolated vertex before the addition or one copy of (p, p) forms a CCB of its own after the addition, or when e_1 and e'_1 exist and belong to the same CCB before the addition. Let F be the face containing p in case of an isolated vertex and the face to the left of e_1 in the other cases. The new halfedges (p, q) and (q, p) are added to the component that contained e_1 (or form a component of their own, if p was an isolated vertex). Call it K. We need to know whether K became non-contractible by the addition. So assume that K was contractible before the addition. We consider any closed cycle in 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 contractible after the addition, contractible before and non-contractible after, or non-contractible before the addition.

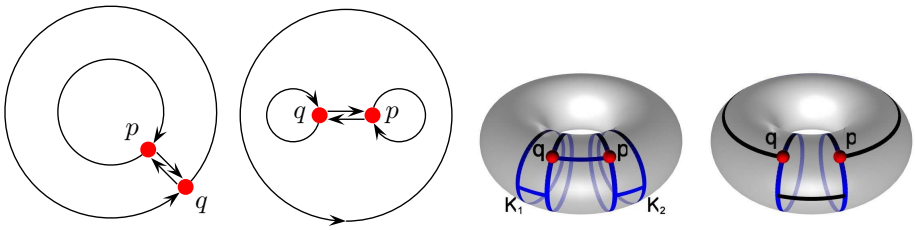


FIGURE 7. Two CCBs are joined into one by the addition of $\{p, q\}$. Four cases can arise. We either join an outer and an inner CCB, or two inner CCBs, or two outer CCBs. In the latter case, the two outer CCBs either belong to distinct components or to the same component.

We discuss the second case in detail, the other cases being simpler. If K stays contractible or K was already non-contractible, F is split 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 an outer boundary, for the face inheriting the type of F, the type (inner, outer) of the CCB is also inherited. We have to assign the new CCBs to F_1 and F_2. If F is disk-like, the assignment is arbitrary. If F is 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. If F is cylinder-like and we split one of its inner CCBs, we locate a point on one of F's outer CCBs with respect to the new CCBs. It will lie to the left of exactly one of them. This CCB becomes an inner CCB of the cylinder-like face. The other new CCB becomes an outer CCB of the disk-like face. We also have to redistribute the inner CCBs 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.

If the component becomes non-contractible, a torus-like F becomes cylinder-like or a cylinder-like F is split into two cylinder-like faces F_1 and F_2. 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 new CCBs become the outer CCBs of F_1 and F_2. We also have to distribute the inner CCBs 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. How do we assign the outer CCBs 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 then 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. We pair the former with B' and the latter with B.

Assume next that e_1 and e_1' belong to distinct CCBs before the addition, say B and B', respectively; see Figure 7. Let K and K' be the components containing them; $K = K'$ is possible. The two CCBs are joined into one. Let F be the face to the left of e_1 and e_1'. We distinguish cases and will 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. 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. 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. 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 now all faces are disk-like. The joined CCB is an outer CCB of F.

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 representing 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 Concept

The refinement hierarchy of the geometry-traits concepts is defined according to the identified minimal requirements imposed by different algorithms that operate on arrangements. The requirements listed by the geometry-traits 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.

The basic concept *OnlyInteriorTraits* suffices to deal with bounded curves in the plane. The deeper levels of the hierarchy (see Figure 8 deal with more complex situations such as objects contained or approaching open, closed, contracted, and identified sides. We use u and v for the coordinates in parameter space, and x, y, z for the coordinates in the ambient space for S.

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 collection of u-monotone and sweepable curves and isolated points. All further operations, presented next, involve only such curves and also points. Those operations serve as the basis for our generalization.

The functions $\mathrm{cmp}_u()$ and $\mathrm{cmp}_v()$ 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 Φ 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, γ denotes a pre-image, that is, $p = \phi_S(u_p, v_p)$ and $C(t) = \phi_S(\gamma(t))$ for all $t \in I$.

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

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 γ, determine whether p is above, below, or lies on C. More precisely, if C is vertical, determine $v_p > v(\gamma(1))$, $v_p > v(\gamma(1))$ or in between. Otherwise, since $u(\gamma(0)) \leq u_p \leq u(\gamma(1))$ and γ is u-monotone, there is a unique $t' \in [0, 1]$ with $u(\gamma(t')) = u_p$. Return $\mathrm{cmp}_v(v_p, \gamma(t'))$. 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 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 to the right of p. More precisely, return $\mathrm{cmp}_v(\gamma_1(\epsilon_1), \gamma_2(\epsilon_2))$, 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.

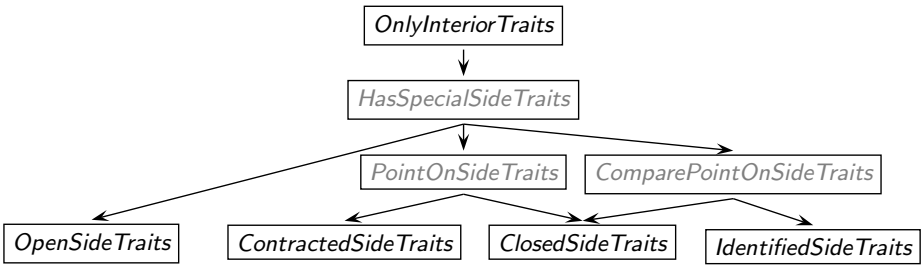


FIGURE 8. Top portion of the refinement hierarchy of the geometry-traits concepts. The grey concepts factor out operations that are common to their descendants. The black concepts correspond to application scenarios. There are four copies of the concepts at the bottom — one fore each side; see also Figure 9 and 10.

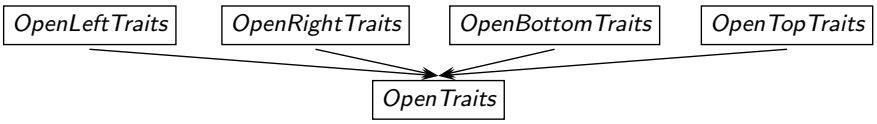


FIGURE 9. Bottom portion of the refinement hierarchy of the geometry-traits concept for curves embedded on an open surface, for instance, the unbounded plane.

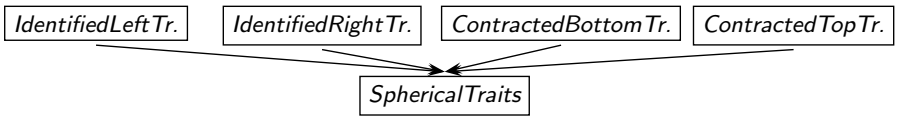


FIGURE 10. Bottom portion of the refinement hierarchy of the geometry-traits concept for curves embedded on a spherical-like parameterized surface.

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

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.

We now come to the bottom level of the hierarchy. For each side of the parameter space there are four concepts: *OpenSideTraits*, *ContractedSideTraits*, *ClosedSideTraits*, and *IdentifiedSideTraits*.[4] This makes for a total of 4^4 concepts. Only a subset of them is meaningful, for instance, if a side is identified, the opposite

[4]Note that we explicitly distinguish closed-only, contracted, and identified sides sides, although they latter two are closed in Φ as well.

side must also be identified, and symmetry reduces the meaningful combinations further. A suitable instantiation for unbounded curves in the unbounded plane is the *OpenTraits* concept shown in Figure 9, while one for surfaces homeomorphic to a sphere can be a model of the combined concept *SphericalTraits* shown in Figure 10.

The four concepts for a given side share operations. They are collected in *HasSpecialSideTraits*, *PointOnSideTraits*, and *ComparePointOnSideTraits* shown grey in Figure 8. The first concept collects the operations required by all refinements and the second and third concept provide operations that handle points, curve-ends, and curves that are contained in the boundary of the parameter domain.

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-axes. More precisely, determine whether $\lim_{t\to d} u(\gamma(t))$ is equal to $u_{\min}$, $u_{\max}$, or falls in between. These predicates determine the location of a curve-end in parameter space.[5]

The next two operations "compare near boundary" determine the order of curve ends lying on the boundary with respect to regular points and among each other. Figure 11 illustrates these comparisons. They return equal if and only if the curves in question lie on top of each other in a neighborhood of the boundary.

Compare u near boundary: There are different predicates for u and v, due to the asymmetry mentioned in Definition 3.1.

(1) Given a regular point p, a curve C, and an index $d \in \{0,1\}$, compare u_p and the "u-coordinate" of the d-end of C. More precisely, return $\mathrm{cmp}_u(p, \gamma(|d-\epsilon|))$, where $\epsilon > 0$ is infinitesimally small.

(2) Given two curves C_1 and C_2 and indices $d_1, d_2 \in \{0,1\}$, compare the "u-coordinates" of the respective ends. More precisely, if the ends are incident to the same horizontal side of the parameter domain, return $\mathrm{cmp}_u(\gamma_1(|d_1-\epsilon_1|), \gamma_2(|d_2-\epsilon_2|))$, where $\epsilon_1, \epsilon_2 > 0$ are infinitesimally small and $v(\gamma_1(|d_1-\epsilon_1|)) = v(\gamma_2(|d_2-\epsilon_2|))$. If they are incident to different horizontal sides, consider the mirror image of one and proceed as above. If only one is incident to a horizontal side, the outcome is clear.

Compare v near boundary: Given two curves C_1 and C_2, and a single index $d \in \{0,1\}$ that identifies two ends of the curves' pre-images γ_1 and γ_2, compare the v-coordinates of two points along γ_1 and γ_2 respectively near the given ends. More precisely, return $\mathrm{cmp}_v(\gamma_1(|d-\epsilon_1|), \gamma_2(|d-\epsilon_2|))$, where $\epsilon_1, \epsilon_2 > 0$ are infinitesimally small and $u(\gamma_1(|d_1-\epsilon_1|)) = u(\gamma_2(|d_2-\epsilon_2|))$.

OpenSideTraits requires one additional predicate:

[5] Note that we only give the definition with respect to the u-dimension, in case the v-version is symmetric. Otherwise, we report additional details.

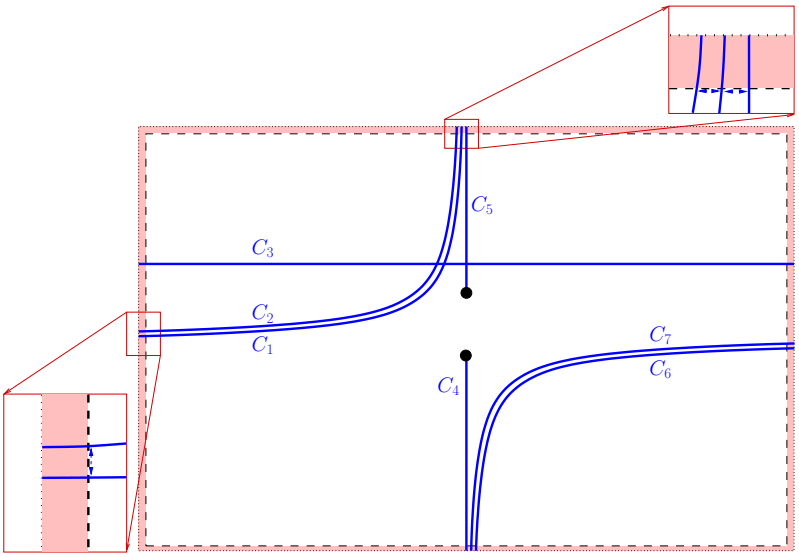


FIGURE 11. Compare curve-ends near boundary in parameter space: We can assume that C_1 and C_7 are given by $xy - 2 = 0$, C_2 and C_6 by $xy - 1 = 0$, C_3 by $y = 2$, C_4 and C_5 are supported by $x = 0$. Then, we have for sufficiently small and suited $\epsilon_i, \epsilon_i' > 0$ the following orders near the boundary. Left side: $v(C_1(\epsilon_1)) < v(C_2(\epsilon_2)) < v(C_3(\epsilon_3))$. Right side: $v(C_6(1-\epsilon_6')) < v(C_7(1-\epsilon_7')) < v(C_3(1-\epsilon_3'))$. Bottom and top side: $u(C_2(\epsilon_2)) < u(C_1(\epsilon_1)) < u(C_4(1-\epsilon_4')) = u(C_4(\epsilon_4)) = u(C_5(1-\epsilon_5')) = u(C_5(\epsilon_5)) < u(C_7(1-\epsilon_7')) < u(C_6(\epsilon_6'))$.

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 γ or not. This predicate tells whether the d-end of C is a point (closed) or open, otherwise.

A closed or contracted sides may contain points. *PointOnSideTraits* introduces one additional predicate:

Locate point in u (resp. v): Given a point p determine its pre-image's location in the domain Φ along the u-dimension. More precisely, check whether u_p is equal to $u_{\min}$ or $u_{\max}$, or falls in between.[6]

ComparePointsOnSideTraits orders points on a boundary lexicographically; it is not needed for contracted sides as they map to a single point on the surface:

[6]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; the pre-image of a point in a closed side is unique.

Compare v on boundary (resp. u): Given points p_1 and p_2 such that $u_{p_1}, u_{p_2} \in \{u_{\min}, u_{\max}\}$, compare v_{p_1} and v_{p_2}.[7]

We are almost done. For a contracted boundary (*ContractedSideTraits*), no additional operation beyond *PointOnSideTraits* is needed. For a closed side, we have to handle curves whose pre-image is fully contained in the boundary of the parameter space. We therefore require:

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

In contrast, identified boundary sides (*IdentifiedSideTraits*) demand a slightly different additional predicate:

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.

5. 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 (open, closed, contracted, identified) of the sides, and the fictitious/non-fictitious distinction for vertices, halfedges and faces. An EDCEL feature (vertex, halfedge, face) *relates* either to the interior or the boundary of the parameter space. A vertex and half-edge 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 half-edges 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 features relating to the boundary.

Obtain boundary kind: Return information on whether a given side of the parameter-space boundary is closed, open, contracted, or identified.

Initialize: Construct an EDCEL that represents an empty arrangement.

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

[7] This function is only used if one of the opposite vertical sides of the boundary is closed or both vertical sides are identified. A point on the identification curve has two pre-images, one for $u_{\min}$ and one for $u_{\max}$. The v-coordinates of both pre-images are the same and so the operation is well-defined.

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 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 feature 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 a 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.

Face split after edge insertion: This function and the next two update CCBs and face types on the insertion of a subcurve such that an inner CCB B

contained in some component K is split into two. This case has been discussed in the paragraph "Maintaining CCBs" of Section 3.2.4 and illustrated in Figure 6. The function determines two Boolean values.

- The first reports whether the face F incident to B splits into two, which is usually the case. The exception is that F is torus-like and the new subcurve makes the contractible component K non-contractible. We check whether a cycle in K containing the new halfedge (p, q) or its twin (q, p) crosses the identification curves an odd number of times.
- The second value is only relevant if F splits. It determines whether K becomes non-contractible. If so, F is cylinder-like.

Is on new face: We are in the situation that a face F is split into faces F_1 and F_2 and that the addition of the edge does not change the type (contractible or non-contractible) of the incident component K. Then, F_1 or F_2 is disc-like. The function determines which of the new CCBs becomes the outer CCB of the new disc-like face.

Boundaries of same face: We are in the situation that a face F is split into faces F_1 and F_2 and that the addition of the edge now makes the affected incident component non-contractible. That is, both F_1 and F_2 are cylinder-like. This function helps to assign the two new outer CCBs to F_1 and F_2 respecting the two existing outer CCBs of F; see Section 3.2.4 for details. More precisely, given one of the old CCBs of F and one of the new CCBs, it checks whether they belong to the same F_i.

Hole creation of edge deletion: This function can be seen as the inverse of the face-split function. In this paper we only discuss the cases when inserting a new curve. Similarly, upon the deletion of a curve being part of an outer CCB and not forming an antenna,[8] the number of components may increase. This must be determined. That is, whether a new inner CCB is introduced.

In addition to these functions, a model of the topology-traits concept also has to also 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.[9]

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.

[8]The curve described by two twin halfedges forms an antenna if both halfedges belong to the same CCB.

[9]The concept also demands for visitors that inserting curves that are pairwise interior disjoint and similar for the previous construction visitor. One can provide either a specialized implementation, or one trivially uses the same as for intersecting curves knowing that their intersection predicate is never invoked.

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.

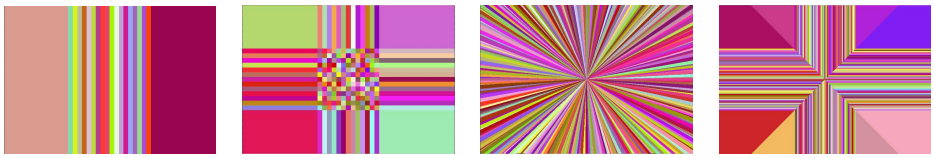
Any topology-traits class is allowed to provide more (surface-specific) functionality — beyond the described minimal concept. An example is to provide methods that enable the access to all vertices lying on an identification curve.

6. Concretizations

Several concretizations of the framework already exist in terms of software. The companion paper [6] 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 plane, which uses an implicit bounding rectangle; see Section 3.2.4 and [31] 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 [5, 14]. 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 [28], and, through a well-known transformation [13], Voronoi diagrams of points in the plane; see [21] 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.

7. 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 arrangement 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



(a) (b) (c) (d)

FIGURE 12. Voronoi diagrams of points in the unbounded plane. (a) Points along a line segment. (b) Points on a grid inside a square. (c) Points on a circle. (d) Points on the boundary of a square.

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 [30] 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, traits classes for different surfaces (see also [6]) 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 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 [25]. 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 achieved 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 [23]. Their 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 for 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

The authors thank Michael Kerber and Ophir Setter for commenting on draft versions of the paper.

References

[1] 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.

[2] 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.

[3] Matthew H. Austern. *Generic Programming and the STL*. Addison-Wesley, 1999.

[4] Jon L. Bentley and Thomas Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, 28(9):643–647, 1979.

[5] Eric Berberich and Pavel Emeliyanenko. CGAL's Curved Kernel via Analysis. Technical Report ACS-TR-123203-04, Algorithms for Complex Shapes, 2008.

[6] Eric Berberich, Efi Fogel, Dan Halperin, Michael Kerber, and Ophir Setter. Arrangements on parametric surfaces II: Concretization and applications, 2009. Submitted to Mathematics in Computer Science.

[7] 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.

[8] 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.

[9] Eric Berberich and Michael Kerber. Exact arrangements on tori and Dupin cyclides. In *Proceedings of ACM Symposium on Solid and Physical Modeling (SPM)*, pages 59–66. Association for Computing Machinery (ACM) Press, 2008.

[10] 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.

[11] Frederic Cazals and Sebastien Loriot. Computing the exact arrangement of circles on a sphere, with applications in structural biology. Technical Report 6049, INRIA Sophia-Antipolis, 2006.

[12] Mark de Berg, Mark van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition, 2000.

[13] Herbert Edelsbrunner and Raimund Seidel. Voronoi diagrams and arrangements. *Discrete & Computational Geometry*, 1:25–44, 1986.

[14] 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).

[15] 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.

[16] Efi Fogel, Dan Halperin, Lutz Kettner, Monique Teillaud, Ron Wein, and Nicola Wolpert. Arrangements. In J.-D. Boissonnat and M. Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, chapter 1, pages 1–66. Springer-Verlag, 2007.

[17] 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.

[18] 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.

[19] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns — Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1999.

[20] 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.

[21] Dan Halperin, Ophir Setter, and Micha Sharir. Constructing two-dimensional Voronoi diagrams via divide-and-conquer of envelopes in space. ACS technical report ACS-TR-361601-01, TAU, 2008.

[22] 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.

[23] Michael Hemmer. *Exact Computation of the Adjacency Graph of an Arrangements of Quadrics*. Ph.D. thesis, Johannes-Gutenberg-Universität, Mainz, Germany, 2008.

[24] 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.

[25] 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.

[26] Kurt Mehlhorn and Stefan Näher. LEDA*: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, UK, 2000.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

Eric Berberich
School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel
e-mail: 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

Dan Halperin
School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel
e-mail: danha@post.tau.ac.il

Kurt Mehlhorn
Max-Planck-Institut für Informatik, Saarbrücken, Germany
e-mail: 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