Sweeping and Maintaining Two-Dimensional Arrangements on Surfaces: A First Step*

Eric Berberich[1], Efi Fogel[2], Dan Halperin[2], Kurt Mehlhorn[1], and Ron Wein[2]

[1] Max-Planck-Institut für Informatik, Saarbrücken, Germany.
{eric,mehlhorn}@mpi-inf.mpg.de
[2] School of Computer Science, Tel-Aviv University, Israel.
{efif,danha,wein}@tau.ac.il

Abstract. We introduce a general framework for sweeping a set of curves embedded on a two-dimensional parametric surface. We can handle planes, cylinders, spheres, tori, and surfaces homeomorphic to them. A major goal of our work is to maximize code reuse by generalizing the prevalent sweep-line paradigm and its implementation so that it can be employed on a large class of surfaces and curves embedded on them. We have realized our approach as a prototypical CGAL package. We present experimental results for two concrete adaptations of the framework: (i) arrangements of arcs of great circles embedded on a sphere, and (ii) arrangements of intersection curves between quadric surfaces embedded on a quadric.

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 the surface S. Arrangements are widely used in computational geometry and have many theoretical and practical applications; see, e.g., [1, 9, 10].

CGAL (`http://www.cgal.org`), the Computational Geometry Algorithms Library aims at a generic and robust, yet efficient, implementation of widely used geometric data structures and algorithms. Until recently, the CGAL arrangement package has dealt only with bounded curves in the xy-plane. This forced users to clip unbounded curves before inserting them into the arrangement; it was the user's responsibility to clip without loss of information. However, this solution is generally inconvenient and outright insufficient for some applications. For example, representing the *minimization diagram* defined by the lower envelope of *unbounded* surfaces in $\mathbb{R}^3$ [13] generally requires more than one unbounded face, whereas an arrangement of bounded clipped curves contains a *single* unbounded face.

* This work has been supported in part by the IST Programme of the EU as Shared-cost RTD (FET Open) Project under Contract No IST-006413 (ACS - Algorithms for Complex Shapes), by the Israel Science Foundation (grant no. 236/06), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

Using the algorithmic principles described in this paper, we have enhanced the arrangement package included in the latest release of CGAL (Version 3.3) to handle unbounded planar curves as well. A more generic version of the package, which handles arrangements embedded on surfaces, is implemented as a prototypical CGAL package.

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 [9]. Mehlhorn and Seel [12] propose a general framework for extending the sweep-line algorithm to handle unbounded curves; however, their implementation can only handle lines in the plane. Arrangements on spheres are covered by Andrade and Stolfi [2], Halperin and Shelton [11], and recently Cazals and Loriot [7]. Cazals and Loriot have developed a software package that can sweep over a sphere and compute exact arrangements of circles on it. Fogel and Halperin [8] exchanged the single arrangement of arcs of great circles on the sphere with six arrangements of linear segments in the plane that correspond to the six faces of a cube circumscribing the sphere. Their approach requires stitching arrangements of adjacent faces. Our approach avoids this overhead. Berberich *et al.* [6] construct arrangements of quadrics by considering the planar arrangements of their projected intersection curves. Their approach requires a postprocessing step (which, however, is not implemented), while our approach avoids the need for a postprocessing step.

Paper outline: In Section 2 we review the Bentley–Ottmann sweep and its implementation in CGAL. We generalize the algorithm in Section 3 to a class of parametric surfaces. After describing the theoretical framework we discuss implementation details in Section 4: in particular, how to encapsulate the topology of the surface into a so-called *topology-traits class*. In Section 5 we give experimental results, and present future-work directions in Section 6.

2 The Bentley–Ottmann Sweep

Recall that the main idea behind the Bentley–Ottmann sweep-line algorithm [3] (and its generalization in [14]) is to sweep over the plane containing a set of bounded curves with a vertical line starting from $x = -\infty$ toward $x = +\infty$, while maintaining the set of x-monotone curves that intersect 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 named the *status structure*. The content of the status structure changes only at a finite number of *events*. The event points are sorted in ascending xy-lexicographic order and stored in an *event queue*. This event queue is initialized with all curve endpoints, and is updated dynamically during the sweep process as new intersection points are discovered. The main sweep process involves the handling of events, namely the insertion of a new curve into the status structure, the removal of a curve that reaches its endpoint, or maintaining the order of intersecting curves. In either case, curves

that become adjacent in the status structure are checked for intersections to the right of the sweep line (having lexicographically greater coordinates) and any such intersection is inserted into the event structure. For our generalization, we call this the main *sweep procedure*. It is preceded by a *preprocessing step* that subdivides, if necessary, the input curves into x-monotone subcurves.

The CGAL implementation of the sweep procedure is generic and independent of the type of curves it handles. All steps of the algorithm are controlled by a small number of geometric primitives, such as comparing two points in xy-lexicographic order, computing intersection points, etc. These primitives are encapsulated in a so-called *geometry-traits class*; see [15] for the full documentation of this concept. Different geometry-traits classes are provided in the arrangement package to handle various families of curves, such as line segments, conic arcs, etc.

The *canonical output* of the sweep-line algorithm consists of the order in which events are processed along with their adjacency information (which events are connected by a curve segment). The implementation in CGAL's arrangement package elegantly decouples the "bare sweep" procedure from the construction of the actual output using *sweep-line visitors* [16]. For example, we use a visitor to convert the canonical output of the sweep procedure into a doubly-connected edge-list (DCEL) representing the arrangement induced by the set of input curves.

3 Sweeping Surfaces

We generalize the sweep to surfaces such as half-planes, cylinders, spheres, tori, etc. and surfaces homeomorphic to these. We describe our generalization in two steps, aiming for an implementation that can handle all these surfaces with maximal code reuse. We capture the geometry of our surfaces through parameterization. A parameterized surface S is defined by a function $f_S : \mathbb{P} \to \mathbb{R}^3$, where $\mathbb{P} = U \times V$ is a rectangular two-dimensional parameter space and f_S is a continuous function. We allow $U = [u_{\min}, u_{\max}]$, $U = [u_{\min}, +\infty)$, $U = (-\infty, u_{\max}]$, or $U = (-\infty, +\infty)$, and similarly for V. Intervals that are open at finite endpoints bring no additional power and we therefore do not discuss them here. For example, the standard planar sweep corresponds to $U = V = (-\infty, +\infty)$, and $f_S(u, v) = (u, v, 0)$. The unit sphere is parameterized via $\mathbb{P} = [-\pi, \pi] \times [-\frac{\pi}{2}, \frac{\pi}{2}]$ and $f_S(u, v) = (\cos u \cos v, \sin u \cos v, \sin v)$.

A *curve* is defined as a function $\gamma : D \to \mathbb{P}$ with (i) D is an open, half-open, or closed interval with endpoints 0 and 1; (ii) γ is continuous and injective, except for *closed curves* where $\gamma(0) = \gamma(1)$; (iii) if $0 \notin D$, the arc has no start point, and emanates "from infinity". It holds $\lim_{t \to 0+} \|\gamma(t)\| = \infty$ (we have a similar condition if $1 \notin D$) and we assume that these limits exist. A *sweepable curve* must, in addition, be weakly u-monotone — that is, if $t_1 < t_2$ then $\gamma(t_1)$ precedes $\gamma(t_2)$ in lexicographic uv-ordering. Consider for example the equator curve on the sphere as parameterized above. This curve is given by $\gamma(t) = (\pi(2t - 1), 0)$, for $t \in [0, 1]$.

We do not assume that either surfaces or curves are given through their parameterization. We use the language of parameterization in our definitions. The algorithm can learn about curves and points only through a well-defined set of geometric predicates provided by the geometry-traits class.

In terms of the standard sweep algorithm, it is convenient to view our sweep as taking place in the parameter space, where we sweep a vertical line from $u_{\min}$ to $u_{\max}$. This is equivalent to sweeping a curve (the image of the vertical line $u = u_0$ under f_S) over the input surface S. Typically, the desired output is embedded on S (e.g., the arrangement induced on S) so one may find it more convenient to view the sweep as taking place over S.

3.1 Bijective Parameterizations and Boundary Events

Recall that the standard Bentley–Ottmann sweep starts by a preprocessing step, which subdivides all input curves into x-monotone subcurves, and initializes the event queue with all their endpoints. When generalizing the algorithm for unbounded curves, we face the problem that these curves do not have finite endpoints. We overcome this difficulty by extending the definition of a *curve-end*, so it may either be a finite endpoint, or represent an unbounded entity in case of a curve that approaches a rim in the parameter space with one of its ends. We say that the curve-end $\langle \gamma, 0 \rangle$ *approaches the west (east) rim* if $\lim_{t \to 0+} \gamma(t) = (-\infty, v_0)$ ($(+\infty, v_0)$, respectively), for some $v_0 \in \mathbb{R} \cup \{-\infty, +\infty\}$, and that it *approaches the south (north) rim* if $\lim_{t \to 0+} \gamma(t) = (u_0, \pm\infty)$ for some $u_0 \in \mathbb{R}$. Thus, in the processing step we associate an event with the two ends of each u-monotone curve $\gamma : D \to \mathbb{P}$. The first (second) event is associated with a finite endpoint if $0 \in D$ ($1 \in D$), and with an unbounded curve-end $\langle \gamma, 0 \rangle$ ($\langle \gamma, 1 \rangle$) if $0 \notin D$ ($1 \notin D$).

In the standard sweep-line procedure, the order of events in the event queue is defined by the xy-lexicographic order of the event points. Here we augment the comparison procedure for two events to handle those events associated with unbounded curve-ends as well. This is done by subdividing the procedure into separate cases, most of which can be handled in a straightforward manner. For example, it is clear that an event approaching the "west rim" is smaller than any event associated with a finite point. When we compare two curve-ends approaching the west rim, we consider the intersection of relevant curves 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). Analogous rules apply to the other situations; see Figure 1(a) for an illustration and [15] for the documentation of the full concept. Note that the rest of the sweep process remains unchanged. In Section 4 we describe how to construct a DCEL that represents an arrangement of unbounded curves.

3.2 Removing Non-Injectivity on the Boundaries

So far we have discussed a simple plane-sweep, with f_S being the identity mapping. When considering more general surfaces, we must handle situations

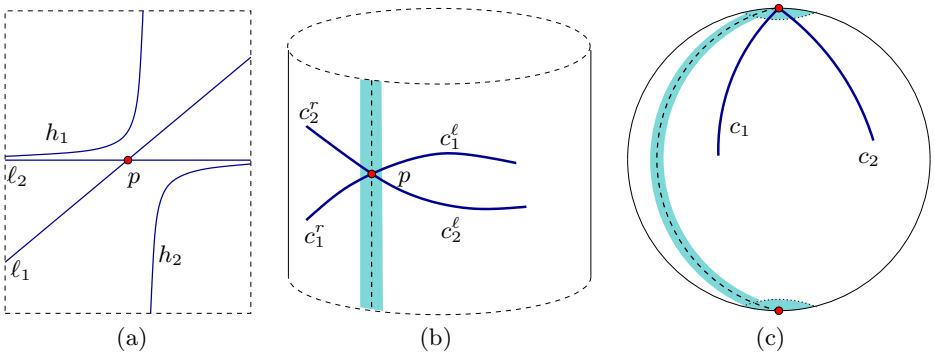
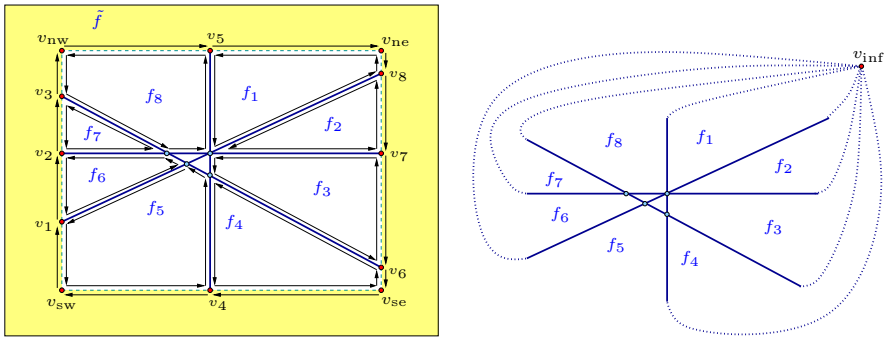


Fig. 1. Comparing sweep events: (a) The order of the events is: left end of ℓ_1, left end of ℓ_2, left end of h_1, right end of h_1, intersection of ℓ_1 and ℓ_2, left end of h_2, right end of h_2, right end of ℓ_2, and right end of ℓ_1. (b) Comparing near the curve of identification: $c_2^\ell < c_1^\ell$ right of the point p and $c_2^r > c_1^r$ to the left of p. (c) Comparing near a contraction point: $c_1 < c_2$.

where the parameterization f_S is not necessarily bijective, so some points in S may have multiple pre-images. If we consider the example of the sphere given in the beginning of this section, then $f_S(-\pi, v) = f_S(+\pi, v)$ for all v, while $f_S(u, -\frac{\pi}{2}) = (0, 0, -1)$ and $f_S(u, \frac{\pi}{2}) = (0, 0, 1)$ for all u. The function $v \mapsto f_S(-\pi, v)$ is a meridian on the sphere, analogous to the "international date line", and the points $(0, 0, \pm 1)$ are the south and the north poles, respectively. The "date line" is induced by the non-injectivity of f_S: as a result, a closed curve on the surface, such as the equator on the sphere, is the image of a non-closed curve in parameter space. The poles also pose a problem: observe that they lie on the sweep line during the entire sweep.

In order to model cylinders, tori, spheres, paraboloids, and surfaces homeomorphic to them, we relax the requirements for the surface parameterizations. First, we require bijectivity only in the *interior* of $\mathbb{P}$ and allow non-injectivity on the boundary (denoted $\partial\mathbb{P}$). More precisely, we require that $f_S(u_1, v_1) = f_S(u_2, v_2)$ and $(u_1, v_1) \neq (u_2, v_2)$ imply $(u_1, v_1) \in \partial\mathbb{P}$ and $(u_2, v_2) \in \partial\mathbb{P}$. On the boundary, we allow injectivity in a controlled way:

- *Contraction* of a side of the parameter space is possible if this side is closed. For example, if $u_{\min} \in U$ and the west rim is contracted, we have $\forall v \in V, f_S(u_{\min}, v) = p_0$ for some fixed point $p_0 \in \mathbb{R}^3$, so the entire west rim is mapped to the same point p_0 on S. We call such a point a *contraction point*. For the sphere, we have contraction at the south and north rims of the parameter space, inducing the south and north poles, respectively.
- *Identification* couples opposite sides of the domain $\mathbb{P}$ and requires each of them to be closed. For instance, the west and east rims of $\mathbb{P}$ are identified if they define the same curve on S, i.e., $\forall v \in V, f_S(u_{\min}, v) = f_S(u_{\max}, v)$. Such a curve is called a *curve of identification*. In our running example of a parameterized sphere, we identify the west and the east rim of the parameter space, and the "international date line" is our curve of identification. A *torus* is modelled by



(a) (b)

Fig. 2. Possible DCEL representations of an arrangement of four lines in the plane.

identifying the two pairs of opposite rims. A *paraboloid* or *cone* can be modelled by identifying the vertical sides of $\mathbb{P}$ and contracting one of the horizontal sides to a point.

We simulate a sweep over $\tilde{S} = f_S(\mathbb{P} \setminus \partial\mathbb{P})$ in the interior of the parameter space. For this, we extend the definition of a *sweepable curve* to be a u-monotone curve whose interior is disjoint from the boundaries of $\mathbb{P}$. Isolated points and curves on $\partial\mathbb{P}$ can be viewed as any of its pre-images. In the preprocessing stage we split the input curves accordingly. In the example shown in Figure 1(b), both input curves c_1 and c_2 cross the line of identification, so each of them is split into two non-closed curved having its endpoints on the two copies of the line of identification. Note that any point lying in $\tilde{S}$ has a unique pre-image in $\mathbb{P}$, and a regular event is generated for it. For curves incident to the boundaries of $\mathbb{P}$ we generate events associated with $\langle\gamma, 0\rangle$ or with $\langle\gamma, 1\rangle$ to indicate γ's ends. As in the unbounded case, we are able to derive the correct order of events using the distinction between normal events that are associated with points on $\tilde{S}$ and events that occur on the boundaries of the parameter space. The same set of additional geometric predicates — namely, comparison of curve-ends on the boundary — is required; see [5] for more details. Similar to the unbounded case, we compare such curve ends in an ε-environment of the boundary; see the examples depicted in Figure 1(b) and (c).

The sweep proceeds exactly as the standard Bentley–Ottmann sweep does. Note however that if we have k sweepable curves incident to a point in $S \setminus \tilde{S}$ (namely a contraction point or a point on the curve of identification), we handle k separate events that relate to this point. In the next section we explain how we "tie all the loose ends" left out by the sweep procedure and construct a well-defined DCEL that represents an arrangement of curves on S.

4 Topology-Traits Classes

As mentioned above, we use a sweep-line visitor to process the topological information gathered in the course of the sweep (the "canonical ouput"), and

construct a DCEL that represents the arrangement of the input curves. While in the case of an arrangement of *bounded* planar curves the DCEL structure is unique and well-defined (see, e.g., [16]), already for unbounded curves in the plane we have a choice of representations, in particular for the representation of the unbounded faces. Figure 2(a) demonstrates one possibility, where we use an implicit bounding rectangle embedded in the DCEL structure using *fictitious* edges that are not associated with any concrete planar curve (this is the representation used in CGAL Version 3.3 [15]). An alternative representation is shown in Figure 2(b). It uses a single *vertex at infinity*, such that all edges that represent unbounded curves are incident to this vertex. Both alternatives are legitimate and each could be more suitable than the other in different situations.

Our implementation aims for flexibility and modularity. For example, we want to give the user the choice between different DCEL-representations as just discussed, and we want to encapsulate the geometric and topological information into a compact interface.

In addition to the *geometry-traits class*, which encapsulates the geometry of the curves that induce the arrangement (see Section 2), we introduce the concept of a *topology-traits class*, which encapsulates the topology of the surface on which the arrangement is embedded. The topology-traits class determines the underlying DCEL representation of the arrangement. In this traits class it is specified whether identifications or contractions take place and whether the parameter space is bounded or unbounded; see [5] for the exact details. Recall that there may be multiple events incident to the same point on the surface boundary. The sweep-line visitor uses the topology-traits class to determine the DCEL feature that corresponds to a curve-end incident to the boundary, so that all these curves will eventually coincide with a single vertex.

For example, if we sweep over the cylinder depicted in Figure 1(b), a vertex w is created on the curve of identification when we insert c_1^ℓ into the arrangement. The topology-traits class keeps track of this vertex, so it will associate w as the minimal end of c_2^ℓ and as the maximal end of c_1^r and c_2^r. Similarly, in the example shown in Figure 1(c), the north pole will eventually be represented as a single DCEL vertex, with c_1 and c_2 incident to it.

We have implemented four topology classes and the corresponding geometry classes: bounded curves in the plane, unbounded curves in the plane, arcs of great circles on a sphere, and intersection curves of quadrics on a quadric. The design of the specialized spherical-topology traits-class pretty much follows the examples we gave throughout the last two sections. We next discuss the topology-traits class for quadric surfaces in more depth.

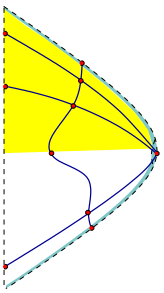
Example: Arrangements of Intersection Curves on a Quadric

A quadratic surface, *quadric* for short, is an algebraic surface defined by the zero set of a trivariate polynomial of degree 2: $\mathcal{Q}(x, y, z) \in \mathbb{Q}[x, y, z]$.We present an implementation of the two-dimensional arrangement embedded on a *base quadric* $\mathcal{Q}_0$, induced by intersections of quadrics $\mathcal{Q}_1, \ldots \mathcal{Q}_n$ with $\mathcal{Q}_0$. Our implementation handles all degeneracies, and is exact as long as the underlying operations are

carried out using exact algebraic number types. We mention that some quadrics (e.g., hyperboloids of two sheets, hyperbolic cones) comprise two connected component. For simplicity, we consider each such component as a separate surface.

Non xy-functional quadratic surfaces can be subdivided into two xy-functional surfaces ($z = f(x, y)$) by a single continuous curve (e.g., the equator subdivides a sphere into two xy-monotone hemispheres), given by $\mathrm{sil}(\mathcal{Q}) = \gcd\left(\mathcal{Q}, \frac{\partial \mathcal{Q}}{\partial z}\right)$. The projection of such a curve onto the xy-plane is called a *silhouette curve*. The projection of an intersection curve between two quadrics onto the xy-plane is a planar algebraic curve of degree at most 4, which can be be subdivided into sweepable curves by intersecting it with $\mathrm{sil}(\mathcal{Q}_0)$. The interior of each such sweepable curve can then be uniquely assigned to the *lower part* or to the *upper part* of the base quadric $\mathcal{Q}_0$. Further details can be found in [6].

In our current implementation we only allow *ellipsoids*, *elliptic cylinders*, and *elliptic paraboloids* for the embedding surface $\mathcal{Q}_0$, as these types are nicely parameterizable by $\mathbb{P} = [l, r] \times [0, 2\pi)$, with $l, r \in \mathbb{R} \cup \{\pm\infty\}$, using $f_S(u, v) = (u, y(v), r(u, y(v), -\sin v))$. We define $y(v) = y_{\min} + (\sin \frac{v}{2})(y_{\max} - y_{\min})$, where $[y_{\min}, y_{\max}]$ denotes the y-range of the ellipse that $\mathcal{Q}_0$ induces on the plane $x = u$. The function $r(x, y, s)$ returns the minimal ($s \leq 0$) or maximal ($s > 0$) value of $\mathcal{R}_{\mathcal{Q},x,y} := \{z \,|\, \mathcal{Q}(x, y, z) = 0\}$, $|\mathcal{R}_{\mathcal{Q},x,y}| \leq 2$. For $\mathcal{Q}_0$, we detect open boundaries or contraction points in u and a curve of identification in v. Note that the quadrics $\mathcal{Q}_1, \ldots, \mathcal{Q}_n$ intersecting the base quadric $\mathcal{Q}_0$ can be arbitrary.



We partition the parameter space of v into two ranges $V_1 := [0, \pi]$ and $V_2 := (\pi, 2\pi)$. Given a point $p(u, v)$ on the base surface $\mathcal{Q}_0$, the *level* of p is $\ell \in \{1, 2\}$ if $v \in V_\ell$ (see the lightly shaded area in the figure to the right, which illustrates this partitioning for an arrangement on a paraboloid). Our geometry-traits class represents a sweepable intersection curve by its projected curve, the level of its projected curve (that needs to be unique in the interior), and its two endpoints. A point $p_i = (u_i, v_i)$ is represented by its projection (x_i, y_i) onto the xy-plane and its level ℓ_i. Given two points p_1 and p_2, their lexicographic order in parameter space is first given by the order of $x_1 = u_1$ and $x_2 = u_2$, and if $x_1 = x_2$ we infer the v-order from (y_1, ℓ_1), (y_2, ℓ_2): if $\ell_1 < \ell_2$ then $p_1 < p_2$, else if $\ell_1 = \ell_2$, then their v-order is identical to their y-order if $\ell_1 = 1$, and opposite to the their y-order if $\ell_1 = 2$.

The topology of an arrangement embedded on a base quadric $\mathcal{Q}_0$ requires special handling. The initialization of the Dcel consists of the creation of a single face, which can be *bounded* or *unbounded* depending on $\mathcal{Q}_0$. For the west and east rim of $\mathbb{P}$, two *boundary vertices*, named v_{left} and v_{right}, are used to record incidences to either points of contraction or to open boundaries. For an ellipsoid both represent points of contraction; for a paraboloid we have one vertex that corresponds to a contraction point and another that represents the unbounded side; for a cylinder, both vertices represent open (unbounded) boundaries. The south and north rims of the parameter space are identified. The topology-traits class maintains a sequence of vertices that lie on the curve of identification, sorted

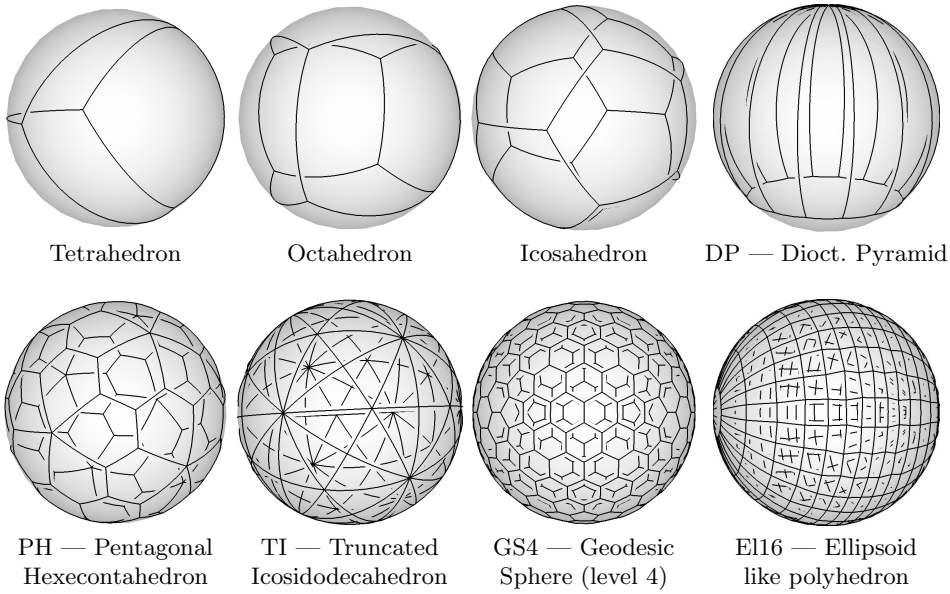


Fig. 3. Gaussian maps of various polyhedra.

by their u-order. This sequence enables the easy identification of different events that correspond to the same point along this curve. We note that comparisons that require geometric knowledge are forwarded to the geometry-traits class, which in turn is implemented using the QuadriX library of Exacus [4].

5 Experimental Results

5.1 Arrangements of Great Arcs on Spheres

We demonstrate the usage of arrangements of arcs of great circles on the unit sphere through the construction of the *Gaussian map* [8] of convex polyhedra, *polytopes* for short. The geometry-traits class defines the point type to be a direction in $\mathbb{R}^3$, representing the place where the a vector emanating from the origin in the relevant direction pierces the sphere. An arc of a great circle is represented by its two endpoints, and by the plane that contains the endpoint directions and goes through the origin. The orientation of the plane determines which one of the two great arcs defined by the endpoints is considered. This representation enables an exact yet efficient implementation of all geometric operations required by the geometry-traits class using *exact rational arithmetic*, as normalizing directions and planes is completely avoided.

The overlay of the Gaussian maps of two polytopes P and Q identifies all the pairs of features of P and Q respectively that have common supporting planes, as they occupy the same space on the unit sphere, thus, identifying all the pairwise features that contribute to the boundary of the *Minkowski sum* of P and Q, namely $P \oplus Q = \{p + q \mid p \in P, q \in Q\}$.

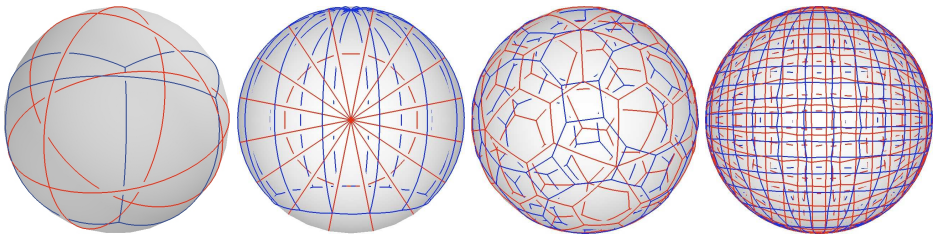


Fig. 4. Gaussian maps of the Minkowski sums.

We have created a large database of models of polytopes. The table below lists, for a small subset of our polytope collection, the number of features in the arrangement of arcs of great circles embedded on a sphere that represents the Gaussian map of each polytope. Recall that the number of vertices (**V**), edges (**E**), and faces (**F**) of the Gaussian map is equal to the number of facets, the number of edges, and the number of vertices in the primal representation, respectively. The table also lists the time in seconds (**t**) it takes to construct the arrangement once the intermediate polyhedron is in place, on a Pentium PC clocked at 1.7 GHz.

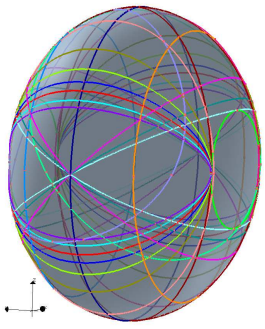
Object	V	E	F	t
Tetrahedron	4	6	4	0.01
Octahedron	6	12	6	0.01
Icosahedron	20	30	12	0.02
DP	17	32	17	0.06
PH	60	150	92	0.13
TI	62	180	120	0.33
GS4	500	750	252	0.56
El16	512	992	482	0.93

We have also conducted a few experiments that construct the Minkowski sums of pairs of various polyhedra in our collection. This demonstrates the successful overlay of pairs of arrangements on a sphere. The table on the right lists the number of features (**V**, **E**, **F**) in the arrangement that represents the Gaussian map of the Minkowski sum and the time in seconds (**t**) it takes to construct the arrangement once the Gaussian maps of the summands are in place. The prefix O before an object name indicates an orthogonal polyhedron. *These performance results are preliminary. We expect the time consumption to reduce significantly once all filtering steps are applied.*

Object 1	Object 2	V	E	F	t
Tetra.	Cube	14	28	16	0.09
DP	ODP	131	130	132	0.63
PH	TI	248	293	340	2.09
El16	OEl16	2260	2290	2322	17.07

5.2 Arrangements on Quadrics

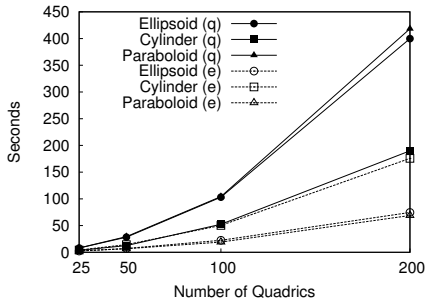
As we mentioned before, our implementation of the quadrics-traits classes is complete, and can handle all kind of degeneracies in a robust manner. The figure on the right shows the arrangement induced by 23 ellipsoids in degenerate position on a base ellipsoid. This highly degenerate arrangement is successfully constructed by our software.



We also measured the performance when computing the arrangement on given base quadrics induced by in-

tersections with other quadrics. As base quadrics we created a random ellipsoid, a random cylinder, and a random paraboloid. These quadrics are intersected by two different families of random quadrics. The first family consists of sets with up to 200 intersecting generic quadrics, sets of the other family include up to 200 ellipsoids intersecting each of the base quadrics. The coefficients of all quadrics are 10-bit integers. All performance checks are executed on a 3.0 GHz Pentium IV machine with 2 MB of cache, with the exact arithmetic number types provided by LEDA (Version 4.4.1).

Table 1 shows the number of arrangement features, as well as time consumption in seconds for selected instances. The figure on the right illustrates the average running time on up to 5 instances containing sets of ellipsoids (e) and general quadrics (q) of different sizes intersecting different base quadrics. Growth is super linear in the number of quadrics, as one expects.



Clearly, the more complex the arrangement, the more time is required to compute it. To give a better feeling for the relative time consumption, we indicate the time spent for each pair of half-edges in the DCEL of the computed arrangement. This time varies in the narrow range between 2.5 ms and 6.0 ms. Other parameters have significant effect on the running time as well, for example the bit-size of the coefficients of the intersection curves.

6 Conclusions and Future Work

We describe a general framework, together with a generic software package, for computing and maintaining 2D arrangements of arbitrary curves embedded on a class of parametric surfaces. We pay special attention to code reuse, which allows the development of traits classes for handling new families of curves and new surface topologies in a straightforward manner. Such developments benefit from a highly efficient code base for the main arrangement-related algorithms.

Single- vs. multi-domain surfaces: In this work we focus on the single domain case, namely our parameter space is represented by a single rectangle: $\mathbb{P} = U \times V$, as described in Section 3. Our major future goal is to extend the framework

Table 1. Performance measures for arrangements induced on three base quadrics by intersections with 50 or 200 quadrics (q), or ellipsoids (e).

Base	**Ellipsoid**				**Cylinder**				**Paraboloid**			
Data	**V**	**E**	**F**	**t**	**V**	**E**	**F**	**t**	**V**	**E**	**F**	**t**
q50	5722	10442	4722	28.3	1714	3082	1370	12.5	5992	10934	4944	29.3
q200	79532	155176	75646	399.8	27849	54062	26214	189.9	82914	161788	78874	418.3
e50	870	1526	658	7.2	1812	3252	1442	14.4	666	1092	428	6.6
e200	10330	19742	9414	74.6	24528	47396	22870	175.8	9172	17358	8189	68.8

to handle general smooth surfaces, which can be conveniently represented by a collection of domains, each of which supported by a rectangular parameter space. The individual 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 topology-traits concept introduced in this paper.

References

1. P. K. Agarwal and M. Sharir. Arrangements and their applications. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 49–119. Elsevier, 2000.
2. M. V. A. Andrade and J. Stolfi. Exact algorithms for circles on the sphere. *Internat. J. Comput. Geom. Appl.*, 11(3):267–290, 2001.
3. J. L. Bentley and T. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Computers*, 28(9):643–647, 1979.
4. E. Berberich, A. Eigenwillig, M. Hemmer, S. Hert, L. Kettner, K. Mehlhorn, J. Reichel, S. Schmitt, E. Schömer, and N. Wolpert. EXACUS: Efficient and exact algorithms for curves and surfaces. In *Proc. 13th ESA*, pages 155–166, 2005.
5. E. Berberich, E. Fogel, D. Halperin, K. Mehlhorn, and R. Wein. A general framework for processing a set of curves defined on a continuous 2D parametric surface, 2007. `http://www.cs.tau.ac.il/cgal/Projects/arr_on_surf.php`.
6. E. Berberich, M. Hemmer, L. Kettner, E. Schömer, and N. Wolpert. An exact, complete and efficient implementation for computing planar maps of quadric intersection curves. In *Proc. 21st SCG*, pages 99–106, 2005.
7. F. Cazals and S. Loriot. Computing the exact arrangement of circles on a sphere, with applications in structural biology. Technical Report 6049, INRIA Sophia-Antipolis, 2006.
8. E. Fogel and D. Halperin. Exact and efficient construction of Minkowski sums of convex polyhedra with applications. In *Proc. 8th ALENEX*, 2006.
9. E. Fogel, D. Halperin, L. Kettner, M. Teillaud, R. Wein, and N. Wolpert. Arrangements. In J.-D. Boissonnat and M. Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, chapter 1, pages 1–66. Spinger, 2006.
10. D. Halperin. Arrangements. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 24, pages 529–562. Chapman & Hall/CRC, 2nd edition, 2004.
11. D. Halperin and C. R. Shelton. A perturbation scheme for spherical arrangements with application to molecular modeling. *Comput. Geom. Theory Appl.*, 10:273–287, 1998.
12. K. Mehlhorn and M. Seel. Infimaximal frames: A technique for making lines look like segments. *J. Comput. Geom. Appl.*, 13(3):241–255, 2003.
13. M. Meyerovitch. Robust, generic and efficient construction of envelopes of surfaces in three-dimensional space. In *Proc. 14th ESA*, pages 792–803, 2006.
14. J. Snoeyink and J. Hershberger. Sweeping arrangements of curves. In *Proc. 5th SCG*, pages 354–363, 1989.
15. R. Wein, E. Fogel, B. Zukerman, and D. Halperin. 2D arrangements. In CGAL-*3.3 User and Reference Manual*. 2007. `http://www.cgal.org/Manual/3.3/doc_html/cgal_manual/Arrangement_2/Chapter_main.html`.
16. R. Wein, E. Fogel, B. Zukerman, and D. Halperin. Advanced programming techniques applied to CGAL's arrangement package. *Comput. Geom. Theory Appl.*, 38(1–2):37–63, 2007.