

A Generic and Flexible Framework for the Geometrical and Topological Analysis of (Algebraic) Surfaces*

Eric Berberich[†]

Max-Planck-Institut für Informatik

Michael Sagraloff[‡]

Max-Planck-Institut für Informatik

Abstract

We present a generic framework on a set of surfaces \mathcal{S} in \mathbb{R}^3 that provides their geometric and topological analysis in order to support various algorithms and applications in computational geometry. Our implementation follows the generic programming paradigm, i.e., to support a certain family of surfaces, we require a small set of types and some basic operations on them, all collected in a model of the newly presented SURFACETRAITS_3 concept. The framework obtains geometric and topological information on a non-empty set of surfaces in two steps. First, important 0- and 1-dimensional features are projected onto the xy -plane, obtaining an arrangement $\mathcal{A}_{\mathcal{S}}$ with certain properties. Second, for each of its components, a sample point is lifted back to \mathbb{R}^3 while detecting intersections with the given surfaces. This idea is similar to Collins' cylindrical algebraic decomposition (cad). In contrast, we reduce the number of liftings using CGAL's Arrangement_2 package as a basic tool. Properly instantiated, the framework provides main functionality required to support the computation of a Piano Mover's instance. On the other hand, the complexity of the output is high, and thus, we particularly regard the framework as key ingredient for querying information on and constructing geometric objects from a small set of surfaces. Examples are meshing of single surfaces, the computation of space-curves defined by two surfaces, to compute lower envelopes of surfaces, or as a basic step to compute an efficient representation of a three-dimensional arrangement.

We also inspire the framework in two steps. First, we show that the well-known family of algebraic surfaces fulfils the framework's requirements. As robust implementations on these surfaces are lacking these days, we consider the framework to be an important step to fill this gap. Second, we instantiate the framework by a fully-fledged model for special algebraic surfaces, namely quadrics. This instantiation already supports main tasks demanded from rotational robot motion planning [Latombe 1993]. How to provide a model for algebraic surfaces of arbitrary degree, is partly discussed in [Berberich et al. 2008].

CR Categories: J.6 [Computer Applications]: Computer-aided Engineering—Computer-aided design (CAD); F.2.2 [Theory of Computation]: Nonnumerical Algorithms and Problems—Geometrical problems and computations;

Keywords: Surface analysis, geometry, topology, framework, generic programming, meshing, space curves, lower envelopes, algebraic surfaces, quadrics

*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).

[†]e-mail: eric@mpi-inf.mpg.de

[‡]e-mail: msagralo@mpi-inf.mpg.de

This is the authors' version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in the *Proceedings of the ACM Solid and Physical Modelling Symposium (SPM 2008)* <http://doi.acm.org/10.1145/1364901.1364925>

1 Introduction

Given a finite set $\mathcal{S} = \{S_1, \dots, S_N\}$ of surfaces in \mathbb{R}^3 , we are interested in the geometric and topological information to describe \mathcal{S} . So, we aim for a decomposition of \mathbb{R}^3 with respect to \mathcal{S} into open cells of dimension 0, 1, 2, and 3. While the first three will be computed explicitly, the 3-dimensional cells are interfaced implicitly. We also want to come up with their adjacency relationship and their position in \mathbb{R}^3 . We go into this direction as various algorithms in computational geometry can be expressed in terms of this information.

Our approach uses projection onto the plane, i.e., its basis consists of a finite arrangement $\mathcal{A}_{\mathcal{S}}$, with some invariant properties for all points of a component of $\mathcal{A}_{\mathcal{S}}$. In particular, they share the same z -pattern when they are lifted. A z -pattern at p encodes the sequence of intersections of $S_i \in \mathcal{S}$ with the vertical line ℓ_p at p . It then suffices to compute a z -pattern only for a sample point of each component of $\mathcal{A}_{\mathcal{S}}$.

In Section 2 we introduce the main problems that must be tackled and present conditions demanded on a family of surfaces to pursue this strategy. We show that all conditions are fulfilled for algebraic surfaces and give an almost minimal arrangement with the required properties.

Section 3 presents our generic implementation. It is an extension of CGAL's Arrangement_2 package. We introduce a new C++-concept which must be implemented for the favored family of surfaces. The concept abstracts the rather complex challenge into a small set of simple tasks demanded on surfaces, like to compute approximations of $S_i \cap \ell_p$. It is the accountability of the framework to combine the output of these operations to obtain the desired output. Some examples how to use the framework in interesting applications are discussed in Section 3.4. Section 4 explains how to implement a model of the concept on the example of quadrics, i.e., algebraic surfaces of degree 2. We conclude with some experimental results.

Related work Our strategy for algebraic surfaces in general follows elimination theory [Basu et al. 2006] and main ideas of cylindrical algebraic decomposition (cad) [Arnon et al. 1984]. A collection of articles emblazing different aspects of cad is given in [Caviness and Johnson 1998]. Many algorithms in computational geometry can be expressed in terms of a cad-instance. A famous example is the Piano Mover's problem that is extensively discussed in [Schwartz et al. 1987]. But unfortunately, many implementations, if any, avoid this technique. We believe for two reasons. One is the quite high complexity of cad. The other is the algebraic focus, that usually requires good knowledge on the topic. Thus, with our framework we want to close the gap, between cad-techniques and implementations of algorithms in computational geometry. Our aim is to provide an easy-to-use framework, with full power on the analysis of surfaces, while always focusing towards applications in computational geometry. As we decouple combinatorics from predicates, it depends on the model used, whether the instantiated framework follows the exact computation paradigm [Yap 1997]. Note that most generic implementations of geometric algorithms show up an undetermined behavior or fail to stop if instantiated with floating-point arithmetic. In the same spirit, we encourage to

use the framework with exact number types and to apply consistent filters for speed-ups.

Two other fields of research are touched by this work. One is the determination of a surface's topology. Several approaches exist, e.g., [Mourrain and T  court 2005; Cheng et al. 2005], but they usually assume a generic coordinate system. Recently, an approach without shearing has been proposed in [Berberich et al. 2008]. Algebraic space curves are discussed in [Gatellier et al. 2005].

The other challenging task is the computation of arrangements of several surfaces. Until now, no complete implementation is available. [Mourrain et al. 2005] presented a method to compute arrangements of quadrics using a space-sweep. An implementation is missing. For two quadrics, a specialized projection approach is available as software [Berberich et al. 2005b]. In contrast to that work, the proposed framework can deal with more than two quadrics, allows more surfaces, and does not pose any generic position assumptions. Thus, it can be interpreted as a key step towards arrangements of surfaces.

2 Problem

Let $\mathcal{S} = \{S_1, \dots, S_N\}$ be a finite set of surfaces in \mathbb{R}^3 . Throughout this section, we detect some conditions that clarify what we consider to be valid surfaces usable in our framework.

For $p = (p_x, p_y)$ a planar point in \mathbb{R}^2 , we denote by $\ell_p \subset \mathbb{R}^3$ the vertical line through p . We tackle the following abstract problems from computational geometry. Abstract means to consider first a surface as set of points.

Problem 2.1. *Given a set of surfaces \mathcal{S} , compute for an arbitrary point $p \in \mathbb{R}^2$ the ordered sequence of intersections of all $S_i \in \mathcal{S}$ with ℓ_p .*

In order to encode the sequence of intersections of $S_i \in \mathcal{S}$, $i = 1, \dots, N$, with ℓ_p we use the following definition of a z-pattern.

Definition 2.2 (z-pattern). *We call the sequence $G_{\mathcal{S},p} = g_1, \dots, g_k$ of subsets of $\{1, \dots, N\}$ a z-pattern with respect to \mathcal{S} and p .*

If we fix p , Problem 2.1 can be split into two, the consecutive Problems 2.3 and 2.7.

Problem 2.3. *Given a surface S compute all intersections of S with ℓ_p .*

We denote $VL_i := \{p \in \mathbb{R}^2 \mid \ell_p \subset S_i\}$ the set of all points $p \in \mathbb{R}^2$ where S_i contains the vertical line ℓ_p .

Condition 2.4. *For a given surface S_i it holds $|VL_i|$ is finite.*

We introduce the following container.

Definition 2.5 (z-stack). *Let $S_i \in \mathcal{S}$, $p = (p_x, p_y) \in \mathbb{R}^2$. A finite subset $Z_{p,i} \subset \{z \in \mathbb{R} \mid (p_x, p_y, z) \in S_i\} \cup \{\pm\infty\}$ is called z-stack of S_i at p if $Z_{p,i}$ contains $\pm\infty$ and $Z_{p,i} = \{\ell_p \cap S_i\} \cup \{\pm\infty\}$ for $p \notin VL_i$. We sort its elements in the following way: $-\infty = z_{p,i,-1} < z_{p,i,0} < \dots < z_{p,i,m_{p,i}-1} < z_{p,i,m_{p,i}} = +\infty$.*

Whereas the container encodes the intersections of a surface S_i with ℓ_p for $p \notin VL_i$ it is intended to store interesting z-coordinates of a surface at a given $p \in VL_i$. Its actual content with respect to $p \in VL_i$ is given by the conditions 2.6 and 2.11 that define how surfaces are allowed to be connected.

We concentrate on the fact, that $m_{p,i}$ denotes its number of finite elements. Obviously, we cannot compute $Z_{p,i}$ for all $p \in \mathbb{R}^2$. Thus, we aim for a subdivision of the plane into finitely many connected cells of dimension 0, 1, and 2 with the property that all points of a cell carry the same m -value. Such a subdivision can be represented as an *arrangement*, where cells are called vertices (0), edges (1), and faces (2). An arrangement is induced by continuous curves and

isolated points, which already define the vertices and edges. Faces are then implicitly defined. More detailed, we aim for surfaces to fulfill the following condition.

Condition 2.6. *Given a surface $S_i \in \mathcal{S}$. Then an arrangement $\mathcal{A}_{\{S_i\}}$ with the following properties exists. $\mathcal{A}_{\{S_i\}}$ consists of a finite number of cells and is induced by a finite number of curves and a finite number of isolated points. $\mathcal{A}_{\{S_i\}}$ contains every point in VL_i as vertex and each cell Γ of $\mathcal{A}_{\{S_i\}}$ is invariant with respect to m , i.e., $\forall p_1, p_2 \in \Gamma : m_{p_1,i} = m_{p_2,i} =: m_{\Gamma,i}$.*

As a consequence, it suffices to only consider a sample point p_Γ of a cell, if one is interested in $m_{p,i}$ for any point of Γ . The container $Z_{p,i}$ also gives geometric information about S_i at p . Let $Z_{\Gamma,i}$ denote the z-stack of an arbitrary sample point p_0 of Γ .

Before we turn to look at how z-stacks are connected we first consider pairs of surfaces. A remaining task to solve Problem 2.1 is the following problem.

Problem 2.7 (Compare entries of z-stacks). *Let $S_i, S_j \in \mathcal{S}$, $i \neq j$. Given a point p and $z_{p,i,u} \in Z_{p,i}$ and $z_{p,j,v} \in Z_{p,j}$ decide whether $z_{p,i,u} < z_{p,j,v}$, $z_{p,i,u} = z_{p,j,v}$, or $z_{p,i,u} > z_{p,j,v}$.*

It is easy to see that the application of a divide-and-conquer strategy that combines Problem 2.3 and 2.7 results in a solution of Problem 2.1. We learn in Section 3.1 that the equality decision is sufficient, as we represent a $z_{p,i,u} \in Z_{p,i}$ with a refineable interval approximation. In the spirit of this problem we define another value.

Definition 2.8 ($m_{p,i,j}$). *Let $S_i, S_j \in \mathcal{S}$, $i \neq j$ and let $p \in \mathbb{R}^2$. Then $m_{p,i,j} := |(Z_{p,i} \cap Z_{p,j}) \setminus \{\pm\infty\}|$. For an $m_{p,i,j}$ -invariant connected set Γ we define $m_{\Gamma,i,j} := m_{p,i,j}$ with a $p \in \Gamma$.*

Again, we can neither compute the intersections of z-stacks nor $m_{p,i,j}$ for an infinite number of points. This fact founds another condition on the surfaces.

Condition 2.9. *Given surfaces $S_i, S_j \in \mathcal{S}$, $i \neq j$. Then an arrangement $\mathcal{A}_{\{S_i, S_j\}}$ exists, with: $\mathcal{A}_{\{S_i, S_j\}}$ consists of a finite number of cells and is induced by a finite number of curves and a finite number of isolated points. $\mathcal{A}_{\{S_i, S_j\}}$ contains every point of $VL_i \cup VL_j$ as a vertex and for each cell Γ of $\mathcal{A}_{\{S_i, S_j\}}$, the following equations hold: $\forall p_1, p_2 \in \Gamma : m_{p_1,i} = m_{p_2,i}, m_{p_1,j} = m_{p_2,j}, m_{p_1,i,j} = m_{p_2,i,j}$.*

Problem 2.10 (Compute planar arrangements). *For given surfaces $S_i, S_j \in \mathcal{S}$, $i \neq j$, compute $\mathcal{A}_{\{S_i\}}$, $\mathcal{A}_{\{S_j\}}$, and $\mathcal{A}_{\{S_i, S_j\}}$.*

It is missing, which connectivity between $Z_{\Gamma_1,i}$ and $Z_{\Gamma_2,i}$, for Γ_1, Γ_2 being incident cells of $\mathcal{A}_{\{S_i\}}$, is allowed:

Condition 2.11. *Let $S_i \in \mathcal{S}$, and $\mathcal{A}_{\{S_i\}}$ an m -regular arrangement. Then S_i is continuous in the following sense:*

1. *Let $p_n \in \Gamma \in \mathcal{A}_{\{S_i\}}$ be a sequence of points with $\lim_{n \rightarrow \infty} p_n = p \in \Gamma^* \in \mathcal{A}_{\{S_i\}}$, and $Z_{p_n,i} = \{z_{p_n,-1}, \dots, z_{p_n,m_{i,\Gamma}}\}$. Then for any $k \in \{-1, 0, \dots, m_{i,\Gamma}\}$ we have $\{\lim_{n \rightarrow \infty} z_{p_n,k} : p_n \in \Gamma \text{ with } p_n \rightarrow p\} = [z_{p,v_k^-}, z_{p,v_k^+}]$ with $z_{p,v_k^\pm} \in Z_{p,i}$ and $\lim_{n \rightarrow \infty} z_{p_n,k} \leq \lim_{n \rightarrow \infty} z_{p_n,l}$ for $k < l$.*
2. *For $p \notin VL_i$ each interval $[z_{p,v_k^-}, z_{p,v_k^+}]$ consists of exactly one point, i.e., $z_{p,v_k^-} = z_{p,v_k^+}$.*

This neighborhood-relationship suffices to encode the connectivity of all lifted cells. We remark that from the above conditions it follows that each cell $\Gamma \in \mathcal{A}_{\{S_i\}} \setminus VL_i$ is the projection of $m_{\Gamma,i} - 2$ connected, disjoint cells of S_i respectively. The lifts of these cells

together with the decomposition of vertical lines ℓ_p into subsegments $(p, [z_{p,v_k^-}, z_{p,v_k^+}])$ form a decomposition of S_i which has the *boundary property*, i.e., the boundary of a cell is given by a union of other cells (compare the similar notion of a CW-complex [Massey 1967], [Bredon 1993]). An analogous formulation also holds for the arrangement $A_{i,j}$ with respect to the surfaces S_i, S_j and their intersection $S_i \cap S_j$.

Whereas the z-stack at a point $p \notin VL_i$ was uniquely determined, its content at points $p \in VL_i$ is only implicitly (not uniquely) given by the chosen arrangement and Condition 2.11.

Problem 2.12 (Compute z-stack). *For given surface $S_i \in \mathcal{S}$ and given point $p \in \mathbb{R}^2$, compute $Z_{p,i}$, even if $p \in VL_i$.*

It remains to compute the connections between lifted cells which is encoded in terms of connections between lifted sample points.

Problem 2.13 (Adjacency). *Given $\mathcal{A}_{\{S_i\}}$ for a surface $S_i \in \mathcal{S}$. Let Γ_1, Γ_2 denote incident cells of $\mathcal{A}_{\{S_i\}}$ and p_1, p_2 their respective planar sample points. Then we are interested in how an entry of $Z_1 := Z_{\Gamma_1,i}$ is connected with the intervals defined by the entries of $Z_2 := Z_{\Gamma_2,i}$. We are asking for a list L of pairs $(a, b) \in A \times B$, with $A := \{0, \dots, m_{\Gamma_1,1} - 1\}$ and $B := \{-1, \dots, m_{\Gamma_2,1}\}$. We distinguish 5 cases for a fixed a_0 :*

$\{b : (a_0, b) \in L\} = \emptyset$: *Indicates, that there exists no continuous path on S_i whose closure connects (p_1, z_{p_1,i,a_0}) with some $(p_2, z_{p_2,i,b}), b \in B$.*

$\{b : (a_0, b) \in L\} = \{b_0\} \wedge b_0 \notin \{-1, m_{\Gamma_2,i}\}$: *The pair (a_0, b_0) then denotes the existence of a continuous path on S_i , lying over Γ_1 , whose closure connects (p_1, z_{p_1,i,a_0}) with (p_2, z_{p_2,i,b_0}) .*

$\{b : (a_0, b) \in L\} = \{b_0\} \wedge b_0 = -1$ ($b_0 = m_{\Gamma_2,i}$): *The pair (a_0, b_0) denotes the existence of a continuous path, lying over Γ_1 , whose closure connects (p_1, z_{p_1,i,a_0}) with the infinite “point” $(p_2, -\infty)$ ($(p_2, +\infty)$), i.e., S_i has a vertical asymptote with respect to z at p_2 .*

$|\{(a_0, b) \in L\}| = 2$: *Let (a_0, b_0) and (a_0, b_1) be these pairs. They denote the existence of an infinite number of continuous paths on S_i , lying over Γ_1 , such that exactly all points $(p_2, z), z \in [z_{p_2,i,b_0}, z_{p_2,i,b_1}]$ are connected with (p_1, z_{p_1,i,a_0}) by considering the closure of a path. Only in case that $b_0 = -1$ or $b_1 = m_{\Gamma_2,i}$, the interval is meant to be open at that end.*

Note that we only compute adjacencies between 0-, 1-, and 2-dimensional cells. The adjacencies to 3-dimensional open cells are given implicitly by them and the projection technique.

We can state our final problem.

Problem 2.14. *Given a set of surfaces \mathcal{S} fulfilling Conditions 2.4, 2.6, 2.9, and 2.11. Compute a finite planar arrangement $\mathcal{A}_{\mathcal{S}}$ with the property that for each of its cells Γ it holds: $\forall p_1, p_2 \in \Gamma : G_{\mathcal{S},p_1} = G_{\mathcal{S},p_2}$. In addition, we want to know how the entries of z-patterns of incident cells in $\mathcal{A}_{\mathcal{S}}$ are connected.*

We claim that our framework implements such a solution for Problem 2.14 given the fact that proper implementations for Problems 2.10 (planar arrangements), 2.12 (z-stacks), 2.7 (compare entries of z-stacks), and 2.13 (adjacency) are provided. The last can be used to derive the desired connectivity of z-pattern entries from the connectivity of z-stacks of single surfaces. In terms of implementation details, we already remark, that implementations for these simpler problems are expected as part of a model that fulfills the

SURFACETRAITS_3 concept. We are newly introducing this concept with all its details in Section 3.1.

We show in Section 2.1 for algebraic surfaces, how to obtain $\mathcal{A}_{\{S_i\}}$ and $\mathcal{A}_{\{S_i, S_j\}}$ without actually computing any z-stack. That section also shows that irreducible algebraic surfaces, our motivating example for the framework, fulfill all stated conditions.

2.1 Algebraic Surfaces

Definition 2.15 (Algebraic surface). *An irreducible algebraic surface $S \subset \mathbb{R}^3$ is defined by the real valued vanishing set of an irreducible polynomial $f = a_n(x, y)z^n + \dots + a_0(x, y) \in \mathbb{R}[x, y, z]$.*

Because of our assumption the polynomial coefficients $a_i(x, y)$ do not share a common component, thus their corresponding planar curves $A_i := V(a_i)$ either intersect in finitely many points or $f = a_0(x, y)$. This follows from Bézout’s Theorem which says that two planar algebraic curves of degree d and e , that do not share a common component, intersect in at most $d \cdot e$ points. Thus the number of common intersection points of A_i is finite, too. It holds for a finite set of irreducible, distinct algebraic surfaces that $\dim(S_i \cap S_j) < 2$, which is a direct consequence of the irreducibility of each surface as two surfaces cannot contain a 2-dimensional component in this case. Under the assumption that Condition 2.9 holds, for $p \notin VL_i$, Condition 2.11 can be verified by the fact that the roots of a polynomial continuously depend on its coefficients, i.e., we must get $\{\lim_{n \rightarrow \infty} z_{p_n,i,-1}, \dots, \lim_{n \rightarrow \infty} z_{p_n,i,m_{i,\Gamma}}\} \subset Z_{p,i}$ and $\lim_{n \rightarrow \infty} z_{p_n,i,k} \leq \lim_{n \rightarrow \infty} z_{p_n,i,l}$ for $k < l$. For $p \in VL_i$ Theorem 2.21 in combination with Definition 2.22 will show that Condition 2.11 also holds in this case.

Now we provide the following (constructive) definition of $\mathcal{A}_{\mathcal{S}}$:

We do not only prove the existence of an arrangement with the desired properties but we will also aim for constructing such an arrangement which is almost minimal with respect to the number of faces, edges and vertices. In order to detect points where the z-pattern changes we need some terms. For two given polynomials $f, g \in \mathbb{R}[x, y, z]$, let $\text{SR}_{i_0}(f, g, z) \in \mathbb{R}[x, y]$ denote their subresultant polynomial and $\text{sr}_{i_0}(f, g, z)$ its leading coefficient (with respect to z). For the exact definitions, as well as for the proof of the Theorem 2.16 we refer to [Basu et al. 2006].

Theorem 2.16. *Let \mathbb{D} be a domain and $f, g \in \mathbb{D}[z]$ be polynomials of degree m and n respectively and $i_0 := \min\{i \mid \text{sr}_i(f, g, z) \neq 0\}$. Then $\text{gcd}(f, g) = \text{SR}_{i_0}(f, g, z)$ and $\deg \text{gcd}(f, g) = i_0$. Especially for $g = f_z = \frac{\partial}{\partial z} f$ it follows that f has a multiple (complex) root if and only if $\text{sr}_0(f, f_z, z) = 0$.*

We remark that $\text{SR}_{i_0}(f, g, z)$ can be obtained as determinant of a Sylvester submatrices. Its entries are expressions in terms of the coefficients of $f \in \mathbb{R}[x, y][z]$. The definition of $\text{SR}_i(f, g, z)$ depends on the degree of f and g with respect to z . For this reason we first have to decompose the plane into components Γ such that for each point $p \in \Gamma$ the polynomial $f(p, z)$ has the same degree. We introduce the following notation.

Definition 2.17. *Let $S = V(f) \subset \mathbb{R}^3$ be a surface of degree n with respect to z . Then we call a point $p = (p_x, p_y) \in \mathbb{R}^2$ m -regular if and only if $m = \max\{i : a_i(p_x, p_y) \neq 0\}$. If $a_i(p_x, p_y) = 0$ for all i , then S contains the vertical line ℓ_p . In this situation we call p (-1) -regular.*

As already mentioned we aim for a decomposition of \mathbb{R}^2 in terms of regularity. This can be achieved by considering the arrangement induced by the planar curves A_i , i.e., the vanishing sets of bivariate polynomials a_i . This arrangement is a decomposition of \mathbb{R}^2 into faces, consisting of n -regular points, and edges and vertices, consisting of points of lower regularity. Notice that it is not necessary to consider the entire arrangement of all A_i : Starting with the arrangement $\mathcal{A}_{\mathcal{S}}^{(0)}$ induced by A_n one

can iteratively define $\mathcal{A}_S^{(j)}$ as the refined arrangement of $\mathcal{A}_S^{(j-1)}$ by inserting the (local 0–dimensional) intersection points of edges $E \subset A_n \cap \dots \cap A_{n-j+1}$ in $\mathcal{A}_S^{(j-1)}$ with A_{n-j} . According to this construction in each step exactly those components are added which are locally extremal with respect to regularity. Thus we finally end up with an arrangement $\mathcal{A}_{\{S\}}^*$ where for each component a certain regularity is allocated. As the S_i do not contain two-dimensional vertical components, all A_i in at most finitely many points, that define VL_i . This shows that Condition 2.4 is fulfilled, too.

For a collection $\mathcal{S} := \{S_1, \dots, S_N\}$ of algebraic surfaces $S_i = V(f_i) = V(a_{i,n_i}(x,y)z^{n_i} + \dots + a_{i,0}(x,y))$ of degree n_i with respect to z , Definition 2.17 has a natural extension.

Definition 2.18. We call a point $p = (p_x, p_y) \in \mathbb{R}^2$ (m_1, \dots, m_N) -regular with respect to $\mathcal{S} := \{S_1, \dots, S_N\}$ if and only if p is m_i -regular with respect to S_i .

Let $S_1, S_2 \in \mathcal{S}$ be two surfaces and $\mathcal{A}_{\{S_1, S_2\}}^*$ the overlay of the arrangements $\mathcal{A}_{S_1}^*$ and $\mathcal{A}_{S_2}^*$, then for each component of $\mathcal{A}_{\{S_1, S_2\}}^*$ the regularity with respect to $\{S_1, S_2\}$ stays invariant. So for each component we can use Theorem 2.16 to get information about the z -patterns $G_{\{S_j\}, p}$, $j = 1, 2$ and $G_{\{S_1, S_2\}, p}$. We will show that there exists a refinement $\mathcal{A}_{\{S_1, S_2\}}^*$ of $\mathcal{A}_{\{S_1, S_2\}}^*$, such that on each component $\Gamma \in \mathcal{A}_{\{S_1, S_2\}}^*$ the z -patterns $G_{\{S_j\}, p}$, $j = 1, 2$, and $G_{\{S_1, S_2\}, p}$ stay the same. For this purpose we have to introduce some further notation.

Definition 2.19. Let $p = (p_x, p_y)$ be an (m_1, m_2) -regular point. Then p has degradation i_0 with respect to $\{S_j\}$ iff $i_0 = \min\{i \mid \text{sr}_i(f_j(p_x, p_y, z), f_{jz}(p_x, p_y, z), z) \neq 0\}$. We say that p has degradation i_{12} with respect to $\{S_1, S_2\}$ iff $i_{12} := \min\{i \mid \text{sr}_i(f_1(p_x, p_y, z), f_2(p_x, p_y, z), z) \neq 0\}$.

Let $\Gamma \in \mathcal{A}_{\{S_1, S_2\}}^*$ be a component of regularity (m_1, m_2) , then there exist common minimal degradations $i_{j,\Gamma}$ and $i_{12,\Gamma}$ for all points on Γ with respect to $\{S_j\}$ and $\{S_1, S_2\}$. More precisely $i_{j,\Gamma} = \min_{p \in \Gamma} \min\{i \mid \text{sr}_i(f_j(p, z), f_{jz}(p, z), z) \neq 0\}$ and $i_{12,\Gamma} = \min_{p \in \Gamma} \min\{i \mid \text{sr}_i(f_1(p, z), f_2(p, z), z) \neq 0\}$.

For a face $\Gamma \in \mathcal{A}_{\{S_1, S_2\}}^*$ all points on Γ have degradations $i_{j,\Gamma} = i_{12,\Gamma} = 0$, except those where $\text{sr}_0(f_j, f_{jz}, z)$ or $\text{sr}_0(f_1, f_2, z)$ vanishes. We denote $\Sigma_j := V(\text{sr}_0(f_j, f_{jz}, z))$, the projected silhouette-curve of S_j and $\Sigma_{1,2} := V(\text{sr}_0(f_1, f_2, z))$ the projected cut-curve of S_1 and S_2 . Now consider the overlay $\mathcal{A}_{\{S_1, S_2\}}^{**}$ of $\mathcal{A}_{\{S_1, S_2\}}^*$ and the planar curves Σ_j and $\Sigma_{1,2}$. Every face of $\mathcal{A}_{\{S_1, S_2\}}^{**}$ consists of points p with the same z -patterns $G_{\{S_j\}, p}$ and $G_{\{S_1, S_2\}, p}$ as a change in one of these patterns can only occur in case of a multiple root of $f_j(p, z)$ or in case of a common root of $f_1(p, z)$ and $f_2(p, z)$.

It remains to refine the edges of $\mathcal{A}_{\{S_1, S_2\}}^{**}$ to get an arrangement with the desired properties. Let $E \in \mathcal{A}_{\{S_1, S_2\}}^{**}$ be an (m_1, m_2) -regular edge with minimal common degradations $i_{j,E}$ and $i_{12,E}$ with respect to $\{S_j\}$ and $\{S_1, S_2\}$. Now passing E , a change in one of the patterns $G_{\{S_j\}, p}$ or $G_{\{S_1, S_2\}, p}$ can only occur at points $p \in E$ where $\text{sr}_{i_{j,E}}(f_j, f_{jz}, z)$ or $\text{sr}_{i_{12,E}}(f_1, f_2, z)$ vanishes. The intersection points of these algebraic curves with E are saved as vertices in the arrangement, which leads to a refined arrangement $\mathcal{A}_{\{S_1, S_2\}}^*$ of $\mathcal{A}_{\{S_1, S_2\}}^{**}$ that has properties as required in Condition 2.9. The construction also shows that we can attach a degradation $i_{j,\Gamma}$ and $i_{12,\Gamma}$ for each component Γ of $\mathcal{A}_{\{S_1, S_2\}}^*$ with respect to $\{S_j\}$ and $\{S_1, S_2\}$.

Observe that by construction every edge $E \in \mathcal{A}_{\{S_1, S_2\}}^*$ is completely contained in one of the curves $V(a_{j,n_j})$, Σ_j or $\Sigma_{1,2}$. However, $\text{sr}_0(f_j, f_{jz}, z)$ contains a factor a_{j,n_j} , thus Σ_j contains $V(a_{j,n_j})$. It follows that the arrangement $\mathcal{A}_{\Sigma_j, \Sigma_{1,2}}$ which is induced by Σ_j and $\Sigma_{1,2}$ already defines the faces of $\mathcal{A}_{\{S_1, S_2\}}^*$. In order to obtain $\mathcal{A}_{\{S_1, S_2\}}^*$ the edges of $\mathcal{A}_{\Sigma_j, \Sigma_{1,2}}$ have to be refined

by adding vertices which result from intersection with coefficient polynomials a_i and higher order subresultants sr_i as shown above.

Definition 2.20. Let $\mathcal{S} = \{S_1, \dots, S_N\}$ and $\mathcal{A}_{\{S_i, S_j\}}$ the arrangement as constructed above. Then we define $\mathcal{A}_{\mathcal{S}}$ to be the overlay of all arrangements $\mathcal{A}_{\{S_i, S_j\}}$.

From our definition, $\mathcal{A}_{\mathcal{S}}$ consists of components $\Gamma \in \mathcal{A}_{\mathcal{S}}$ such that $G_{S,p}$ is identical for all $p \in \Gamma$. It remains to give an exact definition of $Z_{p,i}$ for a surface $S_i \in \mathcal{S}$, which contains the vertical line ℓ_p , and to show that Condition 2.11 is fulfilled in this situation, too. We remark that p is a vertex in $\mathcal{A}_{\mathcal{S}}$ and for a neighborhood of p , none of the surfaces S_j contains a vertical line, except at p . Now we consider a sequence of points $p_n \in \Gamma \subset \mathcal{A}_{\mathcal{S}}$ that converge against p . Then we have to determine possible limits of their k -th lifts $(p_n, z_{p_n, i, k}) \subset p_n \times Z_{p_n, i} \subset S_i$ with respect to S_i . If all p_n lie on an edge E , then the limit is uniquely given as endpoint (above p) of the k -th lift of E with respect to S_i . For a face $F \in \mathcal{A}_{\mathcal{S}}$, adjacent to p , it can happen that the limits of the k -th lifts of two different sequences $p_n, p'_n \in F$ are distinct.

Theorem 2.21. Given a surface $S_i \in \mathcal{S}$, $\Gamma \in \mathcal{A}_{\mathcal{S}}$ and two sequences $p_n, p'_n \in \Gamma$ with $p = \lim_{n \rightarrow \infty} p_n = \lim_{n \rightarrow \infty} p'_n$ and $z_0 = \lim_{n \rightarrow \infty} z_{p_n, i, k}$, $z_1 := z_{p'_n, i, k}$, then for any z^* in between z_0 and z_1 there exists a sequence $p_n^* \in \Gamma$ with $p = \lim_{n \rightarrow \infty} p_n^*$ and $z^* = \lim_{n \rightarrow \infty} z_{p_n^*, i, k}$.

Proof. If $z_0 = z_1$ there is nothing to prove, thus we can assume $z_0 < z_1$. We can further assume that $|p_n - p|$ and $|p'_n - p|$ are monotone. From the definition of $\mathcal{A}_{\mathcal{S}}$ it follows that there exists an ϵ_0 such that $U_\epsilon \cap \Gamma$ is connected for all $\epsilon < \epsilon_0$ and $U_\epsilon := \{q \in \mathbb{R}^2 \mid |q - p| \leq \epsilon\}$. Thus we can assume that $U_n \cap \Gamma$, where $U_n := \{q \in \mathbb{R}^2 \mid |q - p| \leq 2 \max\{|p_n - p|, |p'_n - p|\}\}$, is connected. Now we consider a continuous path $\Pi_n \subset \Gamma \cap U_n$, that connects p_n and p'_n . As the roots of $f(q, z)$ continuously depend on the point $q \in \Gamma$, for each z_n^* in between $z_{p_n, i, k}$ and $z_{p'_n, i, k}$ we can choose a p_n^* that lifts to $z_{p_n^*, i, k} = z_n^*$. As $z_0 = \lim_{n \rightarrow \infty} z_{p_n, i, k}$ and $z_1 := z_{p'_n, i, k}$ there exists an $n_0 \in \mathbb{N}$ such that $z^* \in [z_{p_n, i, k}, z_{p'_n, i, k}]$ for all $n > n_0$. Thus we can choose $z_n^* = z^*$ from which it follows that $p_n^* \in \Gamma$ converges against p and fulfills $z^* = \lim_{n \rightarrow \infty} z_{p_n^*, i, k}$. \square

Theorem 2.21 shows that for any element $\Gamma \in \mathcal{A}_{\mathcal{S}}$, adjacent to p , and any $k \in \{-1, \dots, m_{i,\Gamma}\}$, the set of limits $\lim_{n \rightarrow \infty} z_{p_n, i, k}$ ($p_n \in \Gamma$ a sequence that converge against p) is an interval of $I_{\Gamma, i, k} \subset \mathbb{R}$. Thus defining $Z_{p,i}$ to be the set of all endpoints of these intervals, fulfills Condition 2.11.

Definition 2.22 (z-stack). Let $\mathcal{S} = \{S_1, \dots, S_N\}$ be a set of irreducible algebraic surface and $\mathcal{A}_{\mathcal{S}}$ as defined above. For a surface $S_i \in \mathcal{S}$ the z -stack $Z_{p,i}$ is defined as follows:

- $Z_{p,i} := \{z \in \mathbb{R} \mid (p_x, p_y, z) \in S_i\} \cup \{\pm\infty\}$ for $p \notin VL_i$
- $Z_{p,i} := \{z \in \mathbb{R} \mid \exists \Gamma \in \mathcal{A}_{\mathcal{S}}, k \in \{-1, \dots, m_{i,\Gamma}\} \text{ such that } z \text{ is an endpoint of } I_{\Gamma, i, k}\}$ for $p \in VL_i$

3 Framework

This section reports on details about the implementation of the framework. The algorithmic basis consists of planar arrangements, thus we rely on the matured `Arrangement_2` package [Wein et al. 2007a] of CGAL, the Computational Geometry Algorithm Library.¹ The version shipped with CGAL 3.4 supports several unbounded faces [Berberich et al. 2007]. We omit to repeat how to construct and maintain an arrangement and refer

¹See project homepage: www.cgal.org

to [Agarwal and Sharir 2000] and CGAL’s manual pages. Anyhow, we highlight parts relevant for our purpose. As throughout CGAL, the package implements the generic programming paradigm [Austern 1999], i.e., its actual behavior is determined at compile-time by instantiating template parameters. The important template parameter for our purpose is the `GEOMETRYTRAITS` that has to fulfill CGAL’s `ARRANGEMENTTRAITS_2` concept. The concept defines three types: `Curve_2` that are not necessarily x -monotone, `X_monotone_curve_2` to represent x -monotone sub-curves, and `Point_2` to represent finite endpoints of curves, isolated points, and intersections points. It also defines operations on these types, e.g., to split an instance of type `Curve_2` into `X_monotone_curve_2`s and its isolated points. Other operations are used to construct, maintain, and query arrangements of them. We refer to CGAL’s manual pages for more details on this. The `Arrangement_2` package also provides an incremental construction method (zone-computation) and offers the well-known sweep line paradigm [Bentley and Ottmann 1979] for aggregated constructions as well. An arrangement is maintained as a doubly-connected-edge-list (DCEL) with sophisticated traversals for its components, i.e., vertices, edges, and faces. The package also supports efficient point-location strategies.

Two tools are very important for us. First, it is possible to extend each DCEL-object with a data type. Section 3.2 presents the attached data required for our framework. Second, efficient overlay of arrangements is implemented using the sweep line paradigm, even if the input arrangements are extended by data. A model of the concept `OverlayTraits` has to take care of how to merge data associated to the DCEL-objects of the two originating arrangements [Wein et al. 2007b].

The framework itself is currently part of the SWEEPX library of EXACUS, that is a collection of C++-libraries for Efficient and Exact Algorithms for Curves and Surfaces [Berberich et al. 2005a].² Major parts of EXACUS are currently on the way to be merged into CGAL. In that mood, the framework can be seen as a prototypical CGAL-package, that will be fully integrated in a future release of CGAL. All framework-specific code consists of about 10,000 lines, i.e., not counting CGAL’s arrangement package or other basic classes of the library. Let us first present in Section 3.1 the full concept that a model has to fulfill to use the framework with a new family of surfaces. Sections 3.2 and Section 3.3 give details on how to maintain and to extend planar arrangements to serve our needs. Finally, Section 3.4 exemplary presents concrete applications that emerge from the framework.

3.1 SURFACE TRAITS_3-concept

Our framework must be instantiated with a model of the `SURFACE_TRAITS_3` concept, i.e., a class that implements a collection of syntactic and semantic premises. It does not make assumptions on how to implement them, as long as the demanded functionality is ensured and supported by the formal parameters. A valid model is expected to provide three main types.

`Surface_3`: Used to represent an $S_i \in \mathcal{S}$.

`Arrangement_traits_2`: This type is a model of CGAL’s `ARRANGEMENTTRAITS_2` concept to enable planar arrangements consisting of its `X_monotone_curve_2`s and isolated `Point_2`s. We use it to construct $\mathcal{A}_{\{S_i\}}$ and $\mathcal{A}_{\{S_i, S_j\}}$ and their overlays. It depends on the family of surfaces which model is sufficient. The embedded type `Curve_2` will be used to represent a planar curve, that can be decomposed into 0- and 1-dimensional components.

`Z_at_xy_isolator`: An instance of this type represents and approximates the set $Z_{p,i} \setminus \{\pm\infty\}$ for a given surface S_i at a given `Point_2` p as refineable intervals. The member `number_of_z_coordinates()` gives their number,

i.e., encodes $m_{p,i}$. The values $z = \{\pm\infty\}$ are implicitly handled. `Refineable` means that the z -coordinate with index $0 \leq i < \text{number_of_z_coordinates}()$ might not be known exactly, but at least a lower and an upper boundary is accessible by `lower_boundary(int i)` and `upper_boundary(int i)`. This approximation can be improved by `refine_interval(int i)`. The type of such a boundary is given by `Z_at_xy_isolator::Boundary`.

Besides these types, also some functors are required. We omit to be as precise as a reference manual would, as we want to emphasize the simplicity of the each task.

`Construct_surface_curves_2`: This functor has to provide three operator()s that compute different planar curves emanating from a given surface S . The output is returned as `std::pair< Curve_2, unsigned int >` through an `OutputIterator`. The unsigned `int` defines the multiplicity of a curve, if possible to compute, else -1 is chosen. For example, bivariate polynomials defining algebraic curves can be factorized by multiplicity. The first operator returns all curves that belong to the projection of the silhouette of S , i.e., all 0- and 1-dimensional parts of S , where the z -pattern for some p changes, when slightly moving p away from the projection. The second computes for given m all curves whose points can decrease the regularity of these planar points to m . The last operator compute for given regularity m and given $0 \leq i < m$ all curves whose points can increase a planar point’s degradation to i with respect to the given surface.

`Construct_surface_pair_curves_2`: This functor is very similar to the previous one. It uses the same interface, that here works with respect to two given surfaces `surface1` and `surface2`. Only two operators are demanded from the model. The first operator returns all curves belonging to the projected intersection of the two surfaces. Note that we assumed surfaces to intersect at most 1-dimensional. The second operator returns for given regularity $m_{1,2} = \min(m_1, m_2)$ and given $0 \leq i < m_{1,2}$ all curves whose points can increase a planar point’s degradation to i with respect to the two surfaces.

`Construct_isolator`: Constructs for given point and `surface_i` the correct instance of `Z_at_xy_isolator` type, i.e., it computes $Z_{p,i}$. The given point is a sample point for a cell Γ of $\mathcal{A}_{\{\text{surface}\}}$. To trigger a special or more efficient implementation, integral values carrying the point’s regularity and degradation (or even multiplicity in case of edge) with respect to the given surfaces are interfaced.

`Equal_z`: Checks whether two given intervals of two isolators at a common point are equal, as even in simple cases this decision cannot be finally deduced by iterated refinements of the isolating intervals. Before calling the functor, a set of filters is applied, see Algorithm 3.2 for details. In particular, we know, when called, that all intervals of the two given `isolators` are already refined such, that each interval overlaps with at most one interval of the other isolator. Thus, the set of overlaps forms a candidate list of real intersections. The functor has to decide for the queried candidate, whether there is really an intersection or whether the isolating intervals will separate after a finite number of refinements. Again, the point’s regularities and degradations (and multiplicities in case of an edge) with respect to the given surfaces and the pair of surfaces are interfaced. This may improve the `Equal_z`’s performance, as it usually implements the costly computations, e.g., unavoidable symbolic evaluations in some cases of algebraic surfaces.

`Adjacency`: Given a `Surface_3` with index i , and let Γ_1 and Γ_2 be two given incident cells of $\mathcal{A}_{\{S_i\}}$. For each Γ an instance of type `Z_at_xy_isolator` at respective sample points is also interfaced. The functor is expected to compute the list L as stated in Problem 2.13 represented as `std::list< std::pair< int, int > >`.

This concludes the discussion of the `SURFACE_TRAITS_3` con-

²See project homepage: www.mpi-inf.mpg.de/EXACUS

cept. We strongly encourage to deploy an extensive caching strategy when implementing these functors to avoid unnecessary recomputations.

3.2 Planar arrangements and attached data

The central class of our framework is called `SoX::Projection_2`. It is a reference-counted [Kettner 2006] version of CGAL's `Arrangement_2` instantiated with `Arrangement_traits_2` and using CGAL's `Arr_extended_dcel` to attach an internal data class `SoX::Pdcel_data` to each DCEL-vertex, each DCEL-edge, and each DCEL-face. The class maintains maps to involved surfaces and pairs of surfaces, along with their multiplicities, regularities, and degradations as presented in Section 2. We also store maps to access originating DCEL-handles, after arrangements have been overlaid. Finally, the data class allows to store a sample point and a `Z_stack` for the DCEL-component it is attached to. See Section 3.3 for details on the `Z_stack`. Subsequent operations, as `Construct_isolator` or `Equal_z`, can benefit from this data as first, the global computation save repeated local computations within the functors, and second, the best algorithm according to the given data can be triggered directly. Public members of `Projection_2` also provide access to the stored information for users. As examples we mention `.has_silhouette(Dcel_handle h)` and `.has_cut(Dcel_handle h)`, where `Dcel_handle` is either a vertex-, edge-, or face-handle. In addition, `Projection_2` forwards iterators to traverse all vertices, edges, and faces.

The presented `SURFACETRAITS_3` concept in combination with the generic `Projection_2` class template enables to write generic and reusable code for different families of surfaces. This comprises in particular the construction of arrangements as theoretically presented in Section 2, i.e., to find proper refinements of involved curves and to attach correct multiplicities, regularities, and degradations to each resulting DCEL-component. To do so, we can identify three subtasks, each of which implemented by a functor.

Construct $\mathcal{A}_{\{S\}}$: `Construct_surface_projection_2` provides all curves of interest for that we can easily construct intermediate arrangements with the help of the provided `Arrangement_traits_2` and CGAL's `Arrangement_2`. Using the extension DCEL-components with `Pdcel_data` and proper updates of them, we implemented the iterated construction of $\mathcal{A}_{\{S\}}$ as presented in Section 2 in terms of overlays of the intermediate arrangements and subsequent simplification steps. Our implementation is tuned to only perform the necessary steps, e.g., it stops to assign regularities as soon as all DCEL-features know their value.

Construct $\mathcal{A}_{\Sigma_{i,j}}$: For this task we use the conceptual functor `Construct_surface_pair_projection_2` to first compute an intermediate arrangement consisting of the projected cut-curve only. In a second step, we refine it with respect to $\mathcal{A}_{\{S_i\}}$ and $\mathcal{A}_{\{S_j\}}$ using overlays and subsequent simplifications, i.e., removing features not belonging to $\Sigma_{i,j}$. During these overlays we obtain information about the regularity with respect to $\{S_i\}$ and $\{S_j\}$ of DCEL-components containing $\Sigma_{i,j}$. These are required by the final refinement step, namely to refine the arrangement further with respect to curves that influence the degradation values along $\Sigma_{i,j}$.

Overlay of such arrangements: This task is simple if we are only interested in $\mathcal{A}_{\{S_i, S_j\}}$, as we only have to omit the simplification step in the previous process during the refinement with respect to $\mathcal{A}_{\{S_i\}}$ and $\mathcal{A}_{\{S_j\}}$. Else, $N > 2$ and we have to overlay all $\mathcal{A}_{\{S_i\}}, 1 \leq i \leq s$ and all $\mathcal{A}_{\Sigma_{i,j}}, 1 \leq i < j \leq N$. Let us call them *basic arrangements*. The overlay of curves is directly supported by CGAL's package, which is called inside `Overlay_projection_2`. Its special focus is to take care about the attached data. It is advantageous that different basic arrange-

ment carry independent data, either for a single surface or for a unique pair of surfaces. This observation keeps the merge simple, i.e., we only have to merge the lists of originating key-value-pairs, always distinguishing between single surfaces and pairs of them.

Each functor applies an extensive caching strategies to avoid repeated constructions. This means that for a given surface S_i , there will be exactly one `Projected_2` instance that represents $\mathcal{A}_{\{S_i\}}$, and for $\bar{S} \subset S$ there will be exactly one `Projected_2` instance that represents $\mathcal{A}_{\bar{S}}$.

3.3 Z_stack

We next explain the other important data structure used to represent what happens in the third dimension. From Section 2 it follows that it suffices to compute the z -pattern for a *sample point* of each DCEL-component of \mathcal{A}_S for given S or a subset of it, depending on the desired application. For a vertex, there is no choice for the sample point. For an edge, we have a one-dimensional choice. For a face, the choice is even two-dimensional. For edges and faces, it is useful to construct points with rational coordinates of low bit-size, if possible. The implementation already internally provides methods to construct such sample points. We admit, that the complexity of computed sample points can still be improved in a future version, a task that also depends on the used geometry traits class.

Definition 3.1. A `Z_stack` represents a z -pattern $G = g_1, \dots, g_k$ where each g_i is augmented by (at least) one isolating interval to approximate the meant induced z -coordinate.

In C++, we represent a `Z_stack` by two entities. First, a list of instances of `Z_at_xy_isolator`, i.e., one for each involved surface and second, a sorted container of surface-sets, so-called `Z_cells`. Cells for $\pm\infty$ are not explicitly maintained. A `Z_cell` stores instances of `std::pair< Surface_3, int >`, that we call a *surface-sheet*. Observe that this combination, decouples a z -pattern from the geometric information (i.e., $Z_{p,i}$), but also allows to connect them. A `Z_cell` might store sheets for more than one surface, but all have in common that their corresponding intervals of z -coordinates are isolating with respect to the intervals of the neighboring cells.

Our algorithm to compute a `Z_stack` is basically a merge-sort using Algorithm 3.2 to obtain the order of two cells. The merge-phase is enhanced with filters that are fed by data stored along with the planar DCEL-components.

Algorithm 3.2. Order of two `Z_cells` z_{c1}, z_{c2}

1. Pick a sheet $\langle S_{1,i}, k_{1,i} \rangle$ of z_{c1} and $\langle S_{2,i}, k_{2,i} \rangle$ of z_{c2} .
2. If underlying DCEL-component is a face, proceed with (9).
3. If $\Sigma_{1,2}$ is not involved in DCEL-component, proceed with (9).
4. Refine intervals of two the involved isolators for S_1 and S_2 until each interval overlaps with at most one interval of the other isolator. The overlapping intervals form a positive candidate list for possible intersections of S_1 and S_2 along the current ℓ_p . If no candidate is found, proceed with (9).
5. If DCEL-component is an edge that involves $\Sigma_{1,2}$ and whose multiplicity (degradation) is 1, check whether the single possible overlap matches the cells (i.e., sheets) currently under consideration. If so, return EQUAL, if not, proceed with (9).
6. If DCEL-component is a vertex, select incident edges of $\mathcal{A}_{\{S_1, S_2\}}$ whose `Z_stack` indicate an intersection of S_1 and S_2 . Compute for each `Z_cell` containing an intersection the adjacencies towards given vertex, i.e., a pair of `Z_cells`. If

the pair of `Z_cells` currently in focus is contained in this list, return `EQUAL`, which follows by Condition 2.11. Otherwise proceed with (9).

7. If `DCEL-component` is a vertex, check whether both Σ_1 and Σ_2 are involved. If not, proceed with (9), as only isolated points remain for possible intersections, but intersecting isolated points are indicated by the existence of both projected silhouette-curves.
8. Finally, call `Equal_z` for zc_1 and zc_2 . If it returns `true`, return `EQUAL`.
9. Reaching here indicates that the cells are detected not to be equal, i.e., they can be refined until they do not overlap any more, which gives the correct order, i.e., `SMALLER` or `GREATER`.

We finally want to remark, that the lifting follows the lazy evaluation scheme. This means that sample points for `DCEL-components` and their z -stacks are only computed on demand. Further requests for them are served by cached versions. Of course, `Projection_2` offers public members to access sample points and z -stacks of `DCEL-components`.

When merging attached data due to an overlay, our code always reuses already computed sample points. The same holds for z -stacks and isolators attached to them.

3.4 Applications

The complexity of \mathcal{A}_S with all its z -stacks is quite high due to the projection and the lifting; even for relatively small N . However, the framework can also serve as a key ingredient for concrete applications, e.g., like to compute a surface mesh. To do so, compute a partial vertical decomposition (see [Shaul and Halperin 2002]) of $\mathcal{A}_{\{S\}}$, create its z -stacks along with all adjacency information and lift the resulting faces according to their z -stacks into the third dimension. The situation on their boundaries is specified by proper z -stacks and the computed adjacencies. To obtain a full triangulation, it might be required to split faces further, i.e., with respect to existing crest-lines within a face. The meshing is currently not implemented, in contrast to the following two applications.

Space curves

Definition 3.3. A space curve is the intersection set of two surfaces S_1, S_2 , if at most 1-dimensional.

To represent a space curve, one usually decomposes it into 0- and 1-dimensional parts, where 0-dimensional parts form isolated points, while the 1-dimensional arcs can have properties, like x - or xy -monotonicity. Our implementation provides C++ class templates called `Surface_point_3` and `Surface_arc_3`. The representation of a point is a tuple `(Point_2, Surface_3, int)`, i.e., a planar base point, a supporting surface, and its sheet-number. x - and y -coordinate are given explicitly by the planar point, the z -coordinate is encoded implicitly by the other two types. A bounded 1-dimensional arc in 3D is represented by a tuple `(Surface_point_3, Surface_point_3, X_monotone_curve_2, Surface_3, int, int, int)`, where the points encode the ordered lexicographic smallest and largest point of the arc, and the remaining entries lift the planar curve onto the given surface. The `ints` encode sheet-numbers at the lexicographic smallest and largest point, and in the interior of the arc, where the number must be constant. Note that all three can even be equal or different. Unbounded and vertical arcs are implemented, but omitted in this description.

The following algorithm computes the decomposition of the space-curve defined by S_1, S_2 into isolated vertices and arcs.

Algorithm 3.4. Arcs and points of a space curve

1. Compute $\mathcal{A}_{\{S_1, S_2\}}$ and extract vertices and edges belonging to $\Sigma_{1,2}$.
2. Obtain for each such vertex and each such edge its `Z_stack`.
3. Compute for each such edge the adjacencies towards its source and target vertex (determines how lifted arcs are connected with lifted vertices).
4. For an edge, the `Z_stack` and the adjacencies give all information required to construct instances of type `Surface_arc_3`.
5. Isolated vertices in 3D are detected and constructed by checking whether there exist `Z_cells` over `DCEL-vertices`, that are not adjacent to a `Z_cell` of an edge. It remains to construct proper instances of type `Surface_point_3` via the available information.

Note that vertical arcs are omitted here, as their detection and representation is much simpler.

A careful reader might detect that this approach requires to compute both $\mathcal{A}_{\{S_1\}}$ and $\mathcal{A}_{\{S_2\}}$. Observe, that the output is not demanding for both surfaces at the same time. It suffices to express the decomposition of a space curve into points and arcs only in terms of the surface with *lower complexity*, e.g., the algebraic degree of a surface. Let S_1 be the surface with lower complexity.

We next show how to avoid the computation of $\mathcal{A}_{\{S_2\}}$ and $\mathcal{A}_{\{S_1, S_2\}}$, but at the cost of an additional functor in the `SURFACE-TRAITS_3` concept, called `Common_z`. In contrast to `Equal_z`, which only checks the equality for given intervals, `Common_z` constructs a new instance of type `Z_at_xy_isolator` that represents the common interesting z -coordinates of `Surface_3 surface1` and `Surface_3 surface2` along a vertical line defined by the given `Point_2 point`. Due to lacking $\mathcal{A}_{\{S_2\}}$ and $\mathcal{A}_{\{S_1, S_2\}}$, we do not have access to full knowledge about multiplicities, regularities, and degradations with respect to $\{S_2\}$ and $\{S_1, S_2\}$. Thus, `Common_z` has to deal without these information. It depends on the family of surfaces, how to compute it. We also need a new algorithm to construct the `Z_stack`.

Algorithm 3.5. `Z_stack` for a `DCEL-component` being part of $\Sigma_{1,2}$ within overlay of $\mathcal{A}_{\{S_1\}}$ and $\mathcal{A}_{\Sigma_{1,2}}$

1. Construct `Z_at_xy_isolator isolator1` for S_1 using `Construct_isolator`.
2. Construct `Z_at_xy_isolator isolator12` for intersections of S_1 and S_2 using `Common_z`.
3. Refine intervals of `isolator12` until each is included in an interval of `isolator1`.
4. Create `Z_cell` for each interval of `isolator1` and add S_2 to cell, if interval overlaps with an interval of `isolator12`. Observe that no sheet-number for S_2 is available (but also not needed).

We have implemented this output-sensitive strategy in a class-template called `SoX::Curve_3`. We consider this as a basic implementation that can be used whenever space curves are computed using its projection onto the xy -plane. In this light, this work can be seen as a prototypical implementation of a key ingredient for an upcoming `Curved_kernel_3` in `CGAL`. Section 4 reports on experiments where this approach decomposes intersection curves of quadrics to compute arrangements on quadrics.

Lower envelopes Regard the surfaces in \mathcal{S} as functions in x and y that return for given $p = (p_x, p_y)$ the smallest z -coordinate of the surface's intersections with ℓ_p .

Definition 3.6 (Lower Envelope). *The lower envelope $\mathcal{E}_\mathcal{S}$ of \mathcal{S} is the point-wise minimum of these functions: $\mathcal{E}_\mathcal{S}(x, y) := \min S_i(x, y)$, where the minimum is taken over all functions defined at (x, y) .*

The minimization diagram $\mathcal{M}_\mathcal{S}$ is one way to represent $\mathcal{E}_\mathcal{S}$ as the subdivision of \mathbb{R}^2 into maximal connected cells such that $\mathcal{E}_\mathcal{S}$ is attained by a fixed (possibly empty) subset of functions over the interior of each cell. Meyerovitch presented a generic implementation of a divide-and-conquer algorithm based on CGAL's Arrangement_2 package [Meyerovitch 2006]. The algorithm requires to be instantiated with an extension of CGAL's ARRANGEMENTTRAITS_2 concept. This ENVELOPETRAITS_3 concept additionally demands 3D-types and operations on them. With our framework we provide a generic implementation of such a model, that we are calling `SoX::Surface_3_envelope_traits`.

As types, `Surface_3` and `Xy_monotone_surface_3` are expected. We map both to the `Surface_3` type defined in the used model of the `SURFACETRAITS_3` concept. This may be surprising at first, since a surface in general is not xy -monotone. However, it is only an implementation detail to simplify matters. All later operations that work on an xy -monotone surface S_i consider only the lowest part of the surface. In this light, the required implementation of `Make_xy_monotone_3` is simple. Two construction functors implement the required projections.

`Construct_projected_boundary_2`: Computes for a given S_i of type `Xy_monotone_surface_3` its projected boundary. To provide this information, we compute $\mathcal{A}_{\{S_i\}}$ and traverse its edges vertices. We discard an edge, if its `Z_stack` is empty. If not, compute the `Z_stacks` of its two incident faces. If one stack is empty and the other not, the edge is considered as a projected boundary and returned. The non-empty stack determines on which side of the edge, the surface exists. Vertices are reported if their `Z_stack` contains S_i in their lowest cells, and this cell is not adjacent to a cell of the incident faces' `Z_stacks`.

`Construct_projected_intersection_2`: computes the projected intersection curves of two `Xy_monotone_surface_3s`, called S_i and S_j . We compute the overlay of $\mathcal{A}_{\{S_i, S_j\}}$. Next, we traverse all edges (and isolated vertices) discarding those not participating in $\Sigma_{i,j}$, those with an empty `Z_stack`, and those whose lowest `Z_cell` does not contain S_i and S_j . The remaining edges and vertices are returned.

The concept also requires to implement functors that compare the relative alignment of two `Xy_monotone_surface_3s` in z -direction above a point, above a curve, or above a face incident to a projected intersection curve, that is a sub-face of the projected curve boundaries. Obviously, such a vertical alignment is encoded in a `Z_stack` of the appropriate $\mathcal{A}_{\{S_i, S_j\}}$. Our task is to find the correct one which itself reduces to perform a point location for a given point, or for a constructed point on a curve.

To summarize: Using the this generic model of the `ENVELOPETRAITS_3` concept, computing (lower) envelopes for a family of surfaces boils down to provide a model of the `SURFACETRAITS_3` concept for that class of surfaces. We admit that a specialized model for lower envelopes might be more efficient, but obviously lacks of the possibility to support other (introduced) applications. Section 4 discusses our model of the `SURFACETRAITS_3` concept for quadrics along with initial experiments.

4 The model for quadrics

We implemented in EXACUS' QUADRIX library the model for quadrics that fulfills the `SURFACETRAITS_3` concept.

Definition 4.1. *A quadric is an algebraic surface of degree 2. It is given as the vanishing set of a trivariate polynomial $f(x, y) = a_2(x, y)z^2 + a_1(x, y)z + a_0(x, y)$ in \mathbb{R}^3 .*

We typedef `Surface_3` to QUADRIX's existing class template `QdX::Quadric_3`. Elimination theory gives the following results.

Corollary 4.2. *Let f_1, f_2 be square-free and coprime polynomials defining quadrics $Q_i = V(f_i)$, $i = 1, 2$. The projected silhouette-curve Σ_i of Q_i is given by $\text{res}_z(f_i, f_{i,z}) = \text{sr}_0(f_i, f_{i,z}, z)$ which defines a real plane algebraic curve of degree at most 2. The projected intersection curve of Q_1 and Q_2 , is given by $\text{res}_z(f_1, f_2) = \text{sr}_0(f_1, f_2, z)$ which is a real plane algebraic curve of degree at most 4.*

Our model typedefs `Arrangement_traits_2` to a generic model of CGAL's `ARRANGEMENTTRAITS_2` concept defined in EXACUS's `SWEEPX` library. It is parameterized in the curve-analysis and the analysis of pairs of curves. Instead of the specialized classes for quadrics [Berberich et al. 2005b], we instantiate it with an implementation for algebraic curves of arbitrary degree. The corresponding analysis classes are implemented in EXACUS's `ALCIX` library and details about it can be found in recent publications [Eigenwillig et al. 2007; Eigenwillig and Kerber 2008]. Their main source of efficiency consists in a careful combination of symbolic and approximate computations, without sacrificing the correctness of the overall result. The important reason for this switch is the possibility to approximate a point's x - and y -coordinate as intervals with arbitrary precision. This is crucial, as we want to use the `Bitstream Descartes` method [Eigenwillig et al. 2005] as our `Z_at_xy_isolator` type. This methods isolates the real roots of a polynomial $g(t)$ that is square-free and whose coefficients are "bit-streams", i.e., arbitrary real numbers that are refineable to any positive absolute error. The method itself controls automatically the necessary approximation precision. This exactly serves our goal, as we want to isolate the real roots of $f(p_x, p_y, z) \in \mathbb{R}[z]$ with $p = (p_x, p_y) \notin VL_i$.

The functor `Construct_surface_curves_2` has to provide three operators for a given surface (`Sf_3`). For simplicity we abuse notation and unify surface and its defining polynomial. We also define for $f = \sum_{i=0}^n a_i z^i$ and $m < n$: $f^{(m)} := \sum_{i=0}^m a_i z^i$. The implementation of the functor uses operations on polynomials from EXACUS' `NUMERIX` library to obtain the desired polynomials (see list) and to compute their (cached) square-free factorization. For each factor we construct a cached `std::pair` consisting of an instance of `AcX::Algebraic_curve_2` and an `int` representing its corresponding multiplicity. Each such pair is returned through an `OutputIterator` (`OI`).

```
OI operator()(Sf_3 f, OI oi)
  compute and use  $\text{res}_z(f, f_z)$ 
```

```
OI operator()(Sf_3 f, int m, OI oi)
  obtain and use  $a_m$  with  $0 \leq m \leq 2$ .
```

```
OI operator()(Sf_3 f, int m, int i, OI oi)
  compute and use  $\text{sr}_i(f^{(m)}, f_z^{(m)}, z)$  with  $0 \leq i < m \leq 2$ .
```

For the `Construct_surface_pair_curves_2` functor, exactly the same approach is taken, with the difference that the desired polynomials are expressed with respect to two given surfaces.

```
OI operator()(Sf_3 f1, Sf_3 f2, OI oi)
  compute and use  $\text{res}_z(f_1, f_2)$ 
```

```
OI operator()(Sf_3 f1, int m1, Sf_3 f2, int m2, int i, OI oi)
```

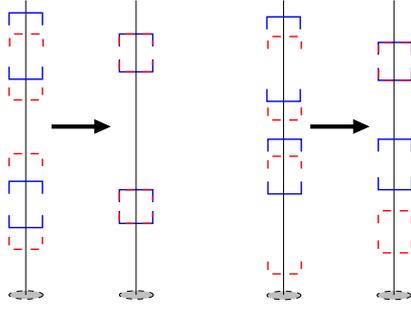


Figure 1: `Equal_z`: Illustration of cases 2 (left) and 3 (right), roots of `Isolator1` are blue, roots of `Isolator2` are red.

compute and use $sr_i(f_1^{(m_1)}, f_2^{(m_2)}, z), 0 \leq i < \min(m_1, m_2) \leq 2$.

These two functors actually are not specific to quadratic algebraic surfaces. They can be used for algebraic surfaces of arbitrary degree, in contrast to the implementations of the other three required functors. We first assume that no vertical line is contained in a quadric.

Lemma 4.3. *The polynomial $f(p_x, p_y, z) \in \mathbb{R}[z]$ has either no real root, a double real root, or two distinct real roots. If it has a double root, then $\text{res}_z(f, f_z)(p_x, p_y) = 0$. Contrary if $\text{res}_z(f, f_z)(p_x, p_y) = 0$ then the z -stack at p contains at most one finite point.*

Proof. The first assertion is rather trivial. If z_0 is a multiple root of $f(p_x, p_y, z)$ then it is also a root of $f_z(p_x, p_y, z)$, thus $\text{res}_z(f, f_z)(p_x, p_y) = 0$. For $a_2(x, y) = 0$ and $\ell_p \not\subset Q$ the backward direction is trivial as in this case $f(x, y, z)$ is a polynomial of degree one or less in z for all (x, y) . If $a_2(x, y) \neq 0$ and $\text{res}_z(f, f_z)(p_x, p_y) = 0$ the polynomials $f(p_x, p_y, z)$ and $f_z(p_x, p_y, z)$ must share a common root z_0 , thus for $\ell_p \not\subset Q$, the z -stack Z_p is given by $\{z_0\} \cup \{\pm\infty\}$. In case where Q contains the vertical line at p , we refer to paragraph about *vertical lines*. \square

`Construct_isolator`: Using Lemma 4.3 we isolate with the Bitstream Descartes method the real roots of $f(p_x, p_y, z)$ in case p 's degradation is 0 or of $f_z(p_x, p_y, z)$ if p 's degradation is greater than 0. Observe that both polynomials fulfill the demanded property of being square-free.

`Equal_z`: Given two surface-sheets and isolators (i.e., `surface1/2`, `isolator1/2`, and `int1/2`). Task: Decide whether the indicated z -approximations are equal. From Algorithm 3.2 we can assume the following when being called. Each interval overlaps with at most one interval or the other isolator and at least one of these overlaps states a true equality. Three cases are left for the two involved quadrics.

1. If some Σ_i exists, the single overlap is the only possible and demanded. Thus return `true`.
2. Both $f_1(x, y, z) = a_2(x, y)z^2 + a_1(x, y)z + a_0(x, y)$ and $f_2(x, y, z) = b_2(x, y)z^2 + b_1(x, y)z + b_0(x, y)$ have two distinct real roots and they are equal at the given p , i.e., there exists a constant $c \in \mathbb{R} \setminus \{0\}$ with $f_1(p_x, p_y, z) = cf_2(p_x, p_y, z)$. This is exactly the case if the two vectors

$$(a_2(p_x, p_y), a_1(p_x, p_y), a_0(p_x, p_y))^T$$

and

$$(b_2(p_x, p_y), b_1(p_x, p_y), b_0(p_x, p_y))^T$$

are linear equivalent, which can be checked by $(a_0b_1 - a_1b_0)(p_x, p_y) = 0 \wedge (a_0b_2 - a_2b_0)(p_x, p_y) = 0 \wedge (a_1b_2 - a_2b_1)(p_x, p_y) = 0$. The checks are implemented as considering $h_{ij} := a_i b_j - a_j b_i$ as planar curves, and test whether p lies on them. This is a basic tool provided by EXACUS' AL-CIX library, which is even filtered by interval arithmetic. If all conditions hold, i.e., two common roots exists (out of two possible), return `true`.

3. Otherwise, two candidate pairs remain for a single equality. We refine them in parallel, until only one candidate is left. If the given values for `root1` and `root2` correspond to that candidate, return `true`, else return `false`.

Adjacency: This functor is implemented mostly combinatorially. If given incident DCEL-components consist of an edge and a vertex, the only possible adjacency is $\langle 0, 0 \rangle$ as the silhouette-curve of a quadric lies in the plane defined by f_z . Assume now that `isolator1` belongs to a face. If $r_1 = 0$, no adjacency can be reported. If $r_1 = 2$, all incident components belong to the Σ . For both $k = 0, 1$ we return $\langle 0, 0 \rangle$. If $r_1 = 1$, the fact that the face's boundary exists allows to conclude by Lemma 4.3 that $r_2 < 0$. This either implies to return $\langle 0, -1 \rangle$ or $\langle 0, 1 \rangle$ to indicate an infinite behavior, or to return $\langle 0, 0 \rangle$, if the quadric is vertical at the sample point of `isolator2`. To distinguish the infinity cases, it is required to obtain the sign of the root of $f_z(p_x, p_y, z) \in \mathbb{R}[z]$, for the sample point p of `isolator1`. Last, if `isolator2` belongs to a face, Lemma 4.3 gives that $r_1 \leq 1$. If $r_1 = 0$, no adjacency exist. Else, three cases are left: (1) $r_2 = 0$, the output consists of no pair (2) $r_2 = 1$, return the pair $\langle 0, 0 \rangle$. (3) $r_2 = 2$, the list contains two pairs, namely $\langle 0, 1 \rangle$, and $\langle 1, 1 \rangle$

Vertical lines A quadric Q contains a vertical line ℓ_p at $p \in \mathbb{R}^2$ exactly if $a_2(x, y) = 0$ and p is an intersection point of the line $L = V(a_1(x, y))$ and the conic $C = V(a_0(x, y))$. Then for each point $(x, y) \notin L$, there exists a unique lift $(x, y, z) \in Q$ with $z = -\frac{a_0(x, y)}{a_1(x, y)}$. Furthermore, there exists no point on Q above any $(x, y) \in L \setminus (L \cap C)$ and for each (of at most 2) intersection point $p \in L \cap C$ the quadric contains the vertical line ℓ_p . The arrangement $\mathcal{A}_{\{Q\}}$ as defined in Section 2 is quite simple in this situation: The projected silhouette-curve L divides \mathbb{R}^2 into two half-planes, which are the faces F_1 and F_2 of $\mathcal{A}_{\{Q\}}$. The intersection points $L \cap C$ represents all vertices in $\mathcal{A}_{\{Q\}}$ and they decompose L into (at most 3) edges. As the edges cannot be lifted onto Q , no adjacency relationship between them and vertices can be reported.

In the following steps we will show how to determine the stack $Z_p = \{-\infty = z_{p,-1}, z_{p,0}, \dots, z_{p,m_p}, z_{p,m_p} = +\infty\}$ and how to get the adjacency information as described in Section 2, condition (4). In Theorem 2.21 we have already proven that for each face $F = F_1$ or $F = F_2$ there exists a corresponding interval I_F such that for each $z^* \in I_v$ we have a sequence $p_n \in F$, converging against p , with $z^* = \lim_{n \rightarrow \infty} z_{p_n} = \lim_{n \rightarrow \infty} -\frac{a_0(p_n)}{a_1(p_n)}$.

From an affine change of coordinates we can assume that $L = V(y)$, i.e., L is the x -axis. Writing $a_0(x, y) = c_0x^2 + c_1y^2 + c_2xy + c_3x + c_4y + c_5$ with variable coefficients $c_i \in \mathbb{R}$, for the z -value of any $(x, y, z) \in Q$ we have

$$z = \frac{c_0x^2 + c_1y^2 + c_2xy + c_3x + c_4y + c_5}{y}$$

For a fixed $y \neq 0$ the set of z -values is given by a parabola ($c_0 \neq 0$), which has its unique local extremum $z_{\max, y}$ at the point x_{\max} with

$$2c_0x_{\max, y} + c_2y + c_3 = 0$$

Thus we get $x_{\max, y} = \frac{-c_3 - c_2y}{2c_0}$ and

$$z_{\max, y} = \frac{4c_0c_5 + 4yc_0c_4 - 2yc_2c_3 - c_3^2 + 4y^2c_0c_1 - y^2c_2^2}{4yc_0}$$

Now we distinguish three cases:

1. $c_0 = 0$: C and L intersect in a unique point $p = (-\frac{c_5}{c_3}, 0)$.
Now given an arbitrary $z^* \in \mathbb{R}$, we have

$$f(x_y, y, z^*) = 0 \Leftrightarrow x_y(c_2y + c_3) = yz^* - c_1y^2 - c_4y - c_5$$

For $y \rightarrow 0$ we get $x_y \rightarrow -\frac{c_5}{c_3}$, thus the conic C_{z^*} , which is implicitly given by the equation $x(c_2y + c_3) = yz^* - c_1y^2 - c_4y - c_5$, passes the point p and the face F . It follows the existence of a sequence $p_n \rightarrow p \in V(a_0) \cap V(a_1)$ with $z_{p_n} \rightarrow z^*$.

2. $|V(a_0) \cap V(a_1)| = 0$ or 2 and $c_0 \neq 0$: For a fixed $z^* \in \mathbb{R}$, the conic $C_{z^*} = V(f(x, y, z^*))$ has exactly two intersection points with L . Hence, C_{z^*} has only ordinary intersections with L . Thus, C_{z^*} contains an arc that passes $V(a_0) \cap V(a_1)$ and the face F . It follows the existence of a sequence $p_n \rightarrow p \in V(a_0) \cap V(a_1)$ with $z_{p_n} \rightarrow z^*$.
3. $|V(a_0) \cap V(a_1)| = 1$: In this case the quadratic polynomial $a_0(x, 0)$ has a multiple root and $p = V(a_0) \cap V(a_1) = (\frac{-c_3}{2c_0}, 0)$. Hence, we get $c_3^2 - 4c_5c_0 = 0$ and furthermore

$$\begin{aligned} \lim_{y \rightarrow 0} z_{\max, y} &= \lim_{y \rightarrow 0} \frac{4yc_0c_4 - 2yc_2c_3 + 4y^2c_0c_1 - y^2c_2^2}{4yc_0} \\ &= \frac{2c_0c_4 - c_2c_3}{2c_0} \end{aligned}$$

Now, the line L_{\max} , implicitly given by $2c_0x + c_2y + c_3 = 0$ passes the point p and contains a sequence of points $p_n \rightarrow p$ with $z_{p_n} \rightarrow \frac{2c_0c_4 - c_2c_3}{2c_0} =: z_{p,0}$ for $n \rightarrow \infty$. In the next step we will show that for any other sequence $p'_n \rightarrow p$ we must get $z_{p'_n} \rightarrow z' \geq z_{p,0}$ or $z' \leq z_{p,0}$ depending on whether $c_0 > 0$ or $c_0 < 0$. W.l.o.g. we assume that $c_0 > 0$. Then for a fixed y the parabola $\frac{c_0x^2 + c_1y^2 + c_2xy + c_3x + c_4y + c_5}{y}$ has a global minimum $z_{\max, y}$ at $x_{\max, y}$. Thus, for any point $p'_n = (x'_n, y'_n)$ we must have $z_{p'_n} = -\frac{a_0(x'_n, y'_n)}{a_1(x'_n, y'_n)} \geq z_{\max, y'_n}$. It follows that $\lim_{p'_n \rightarrow p} z_{p'_n} \geq \lim_{p'_n \rightarrow p} z_{\max, y'_n} = z_{p,0}$.

In the two cases (1) and (2) we have shown that the lifts of the two faces F_1 and F_2 are both adjacent to any point on the vertical line ℓ_p , i.e., for any $(p, z^*) \in \ell_p$ there exist sequences $p_n^{(j)} \in F_j$ with $(p_n^{(j)}, z_{p_n^{(j)}}) \rightarrow (p, z^*)$ for $j = 1, 2$. Thus, it suffices that $Z_p = \{\pm\infty\}$, i.e., `ConstructIsolator` returns an empty instance and so `Equal_z` is trivial. It is clear that `Adjacency` returns for an incident face F towards p the pairs $\langle 0, -1 \rangle$ and $\langle 0, 0 \rangle$.

In the third case, for exactly all z^* in between $\frac{2c_0c_4 - c_2c_3}{2c_0}$ and $\pm\infty$ (depending on whether $c_0 > 0$ or $c_0 < 0$, respectively) there exists a sequence $p_n^{(j)} \in F_j$ with $p_n^{(j)} \rightarrow p$ and $z_{p_n^{(j)}} \rightarrow z^*$. As we can also pursue the affine coordinate transformation in case where f is given with arbitrary variable coefficients, it is possible to get formulas in terms of these coefficients to decide in which case we are and to determine the single non-infinity entry $z_{p,0}$ of Z_p . Observe that $p = (p_x, p_y)$ and $z_{p,0}$ are all rational. Thus, `Equal_z` can be implemented in terms of rational arithmetic. `Adjacency` returns for an incident face towards the vertical line at p , either the pairs $\langle 0, -1 \rangle$, $\langle 0, 0 \rangle$ or the pairs $\langle 0, 0 \rangle$, $\langle 0, 1 \rangle$, depending on the sign of c_0 and the face.

#Surfaces	#DCEL	#z-cells	t	t/cell
2 ellipsoids	13	12	0.1s	5.7ms
4 ellipsoids	230	904	2.8s	3.4ms
6 ellipsoids	877	5942	19.9s	3.7ms
8 ellipsoids	2780	25220	171.9s	7.2ms
10 ellipsoids	4952	52788	582.0s	11.5ms
2 quadrics	53	160	0.4	2.7ms
4 quadrics	1099	7172	19.7	3.0ms
6 quadrics	3946	39254	194.4	5.4ms
8 quadrics	9983	132352	2306.1	18.1ms

Table 1: Typical performance measures for sets of ellipsoids and arbitrary quadrics of increasing size.

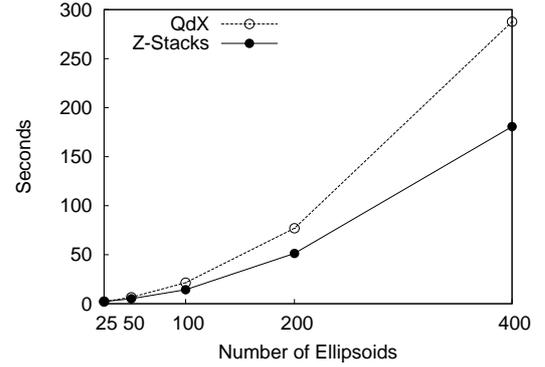


Figure 2: Running times to compute arrangements on an ellipsoid intersected by other ellipsoids.

Results

We tested the performance of the framework for quadrics by computing all z -stacks and all adjacencies for \mathcal{A}_S , where $|\mathcal{S}|$ increases. All experiments are executed on a Pentium IV CPU with 3.0 GHz clock-speed and 2 MB of cache. The executables are compiled with gnu's C++-compiler in version 3.3 with disabled debugging (`-DNDEBUG`) and enabled optimizations (`-O2`) and the exact number types of LEDA. Table 1 lists example runs.

It cannot be hidden, that $|\mathcal{S}|$ seems quite small, but on the other hand, the size of the output grows rapidly. For 8 quadrics we already have to compute nearly 10.000 z -stacks containing more than 130.000 cells, numbers much larger than the ones for two surfaces. On the contrary, the time spent per cell grows much slower. In fact, we have to see an increasing number here, as by construction of the data, similar intervals along each ℓ_p are intersected by a growing number of quadrics, i.e., it requires additional time to isolate the cells against each other. These first experiments lead to the recommendation to use the framework especially for cases where $|\mathcal{S}|$ is small, like in the next example, where only pairs or triples of surfaces are needed.

We also tested the framework when computing arrangements on a reference quadric induced by intersections with other quadrics. As example we chose increasing sets of random ellipsoids. Figure 2 shows the improvement of about 30%, compared to a former implementation [Berberich et al. 2007]. For general quadrics the ratio is similar. We note that an example arrangement induced by 400 ellipsoids intersecting the reference ellipsoids consists of about 38.000 vertices and 74.000 edges, which is now possible to compute in around 180s instead of 287s. These results imply to analyze the performance of the framework with more extensive experiments.

5 Conclusion and outlook

We presented a generic framework, implemented in C++, based on CGAL's `Arrangement_2` package to compute z -stacks of (algebraic) surfaces and adjacencies between z -stacks. The design of it is simple, the interface intuitive and the approach taken does not enforce to assume generic position. A new family of surfaces can be used by implementing a small set of tasks collected in a newly introduced `SURFACE_TRAITS_3` concept. Our first model provides the full functionality for quadrics, i.e., algebraic surfaces of degree 2 being developed within the EXACUS-project [Berberich et al. 2005a]. As the number of z -cells grows fast, we see the main application of this tool in providing information for a small set of surfaces, i.e., to compute the topology (and geometry) of a single surface, a single space-curve, or to serve as a key ingredient for high-level algorithms like the computation of envelopes, or three-dimensional arrangements. Some of them are already presented and implemented, others are open for future research, in particular, how to reduce the complexity for certain applications. The next goal is to negotiate in detail, how to query the framework instantiated with quadrics to support rotational robot motion planning, which is an intermediate step towards a full support for Piano Mover's instances.

Besides the applications there are also more framework-related open problems. It would be beneficial to simplify the conditions. This should enable to model other families of surfaces as well. In addition, it helps to describe the individual steps more generically. We are already working to release the binding to algebraic surfaces and to come up with a much more generic version.

Concerning models, the current challenge to solve is to provide a full model of our `SURFACE_TRAITS_3` concept for algebraic surfaces of arbitrary degree. This model should efficiently implement all the simple operations. A clever combination of cheap approximative and unavoidable costly symbolic computations together with the full set of modular, floating-point-, modular- and geometric filters is promising to be successful. The case of a single surface is discussed in [Berberich et al. 2008]. If completed, another goal is to implement a model for surfaces composed of patches of algebraic surfaces.

Acknowledgements We thank Michael Kerber for hours of fruitful discussions and for giving comments on a preliminary version of this paper.

References

- AGARWAL, P. K., AND SHARIR, M. 2000. Arrangements and their applications. In *Handbook of Computational Geometry*, J.-R. Sack and J. Urrutia, Eds. Elsevier, 49–119.
- ARNON, D. S., COLLINS, G. E., AND MCCALLUM, S. 1984. Cylindrical algebraic decomposition i+ii. *SIAM Journal on Computing* 13, 865–889. Reprinted in [Caviness and Johnson 1998], pp. 136–165.
- AUSTERN, M. H. 1999. *Generic Programming and the STL*. Addison-Wesley.
- BASU, S., POLLACK, R., AND ROY, M.-F. 2006. *Algorithms in Real Algebraic Geometry*, 2nd ed., vol. 10 of *Algorithms and Computation in Mathematics*. Springer.
- BENTLEY, J. L., AND OTTMANN, T. A. 1979. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers* C-28, 643–647.
- BERBERICH, E., EIGENWILLIG, A., HEMMER, M., HERT, S., KETTNER, L., MEHLHORN, K., REICHEL, J., SCHMITT, S., SCHÖMER, E., AND WOLPERT, N. 2005. Exacus: Efficient and exact algorithms for curves and surfaces. In *Proceedings of the 13th Annual European Symposium on Algorithms (ESA 2005)*, Springer, vol. 3669 of *LNCS*, 155–166.
- BERBERICH, E., HEMMER, M., KETTNER, L., SCHÖMER, E., AND WOLPERT, N. 2005. An exact, complete and efficient implementation for computing planar maps of quadric intersection curves. In *21st Annual Symposium on Computational Geometry (SCG'05)*, ACM, Pisa, Italy, J. Mitchell, G. Rote, and L. Kettner, Eds., Association for Computing Machinery (ACM), 99–106.
- BERBERICH, E., FOGEL, E., HALPERIN, D., MEHLHORN, K., AND WEIN, R. 2007. Sweeping and maintaining two-dimensional arrangements on surfaces: A first step. In *ESA*, Springer, L. Arge, M. Hoffmann, and E. Welzl, Eds., vol. 4698 of *Lecture Notes in Computer Science*, 645–656.
- BERBERICH, E., KERBER, M., AND SAGRALOFF, M. 2008. Exact geometric-topological analysis of algebraic surfaces. In *Proceedings of the 24th Annual Symposium on Computational Geometry (SCG 2008)*. to appear.
- BREDON, G. E. 1993. *Topology and Geometry*. Springer.
- CAVINESS, B. F., AND JOHNSON, J. R., Eds. 1998. *Quantifier Elimination and Cylindrical Algebraic Decomposition, Texts and Monographs in Symbolic Computation*. Springer.
- CHENG, J.-S., GAO, X.-S., AND LI, M. 2005. Determining the topology of real algebraic surfaces. In *IMA Conference on the Mathematics of Surfaces*, Springer, R. R. Martin, H. E. Bez, and M. A. Sabin, Eds., vol. 3604 of *Lecture Notes in Computer Science*, 121–146.
- EIGENWILLIG, A., AND KERBER, M. 2008. Exact and efficient 2d-arrangements of arbitrary algebraic curves. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA08)*, 122–131.
- EIGENWILLIG, A., KETTNER, L., KRANDICK, W., MEHLHORN, K., SCHMITT, S., AND WOLPERT, N. 2005. A Descartes algorithm for polynomials with bit-stream coefficients. In *8th International Workshop on Computer Algebra in Scientific Computing (CASC 2005)*, vol. 3718 of *LNCS*, 138–149.
- EIGENWILLIG, A., KERBER, M., AND WOLPERT, N. 2007. Fast and exact geometric analysis of real algebraic plane curves. In *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation (ISSAC 2007)*, C. W. Brown, Ed., 151–158.
- GATELLIER, G., LABROUZY, A., MOURRAIN, B., AND TÉCOURT, J.-P. 2005. Computing the topology of 3-dimensional algebraic curves. In *Computational Methods for Algebraic Spline Surfaces*, Springer, 27–44.
- KETTNER, L. 2006. Reference counting in library design—optionally and with union-find optimization. In *Proceedings of the First International Workshop on Library-Centric Software Design, LCSD'05*, Rensselaer Polytechnic Institute, Computer Science Department, San Diego, CA, USA, D. Musser and J. Siek, Eds., vol. 06-12 of *Technical Report*, 34–43.
- LATOMBE, J.-C. 1993. *Robot motion planning*, 3rd printing ed., vol. 0124; of *Kluwer international series in engineering and computer science; SECS*. Kluwer.
- MASSEY, W. 1967. *Algebraic Topology: An Introduction*. Springer.

- MEYEROVITCH, M. 2006. Robust, generic and efficient construction of envelopes of surfaces in three-dimensional space. In *Proceedings of 14th European Symposium on Algorithms (ESA)*, vol. **4168** of LNCS, 792–803.
- MOURRAIN, B., AND TÉCOURT, J.-P. 2005. Isotopic meshing of a real algebraic surface. Technical Report 5508, INRIA Sophia-Antipolis.
- MOURRAIN, B., TÉCOURT, J.-P., AND TEILLAUD, M. 2005. On the computation of an arrangement of quadrics in 3d. *Comput. Geom. Theory Appl.* 30, 2, 145–164.
- SCHWARTZ, J. T., SHARIR, M., AND HOPCROFT, J. E., Eds. 1987. *Planning, geometry, and complexity of robot motion*. Ablex series in artificial intelligence. Ablex Publ.
- SHAUL, H., AND HALPERIN, D. 2002. Improved construction of vertical decompositions of three-dimensional arrangements. In *SCG '02: Proceedings of the eighteenth annual symposium on Computational geometry*, ACM, New York, NY, USA, 283–292.
- WEIN, R., FOGEL, E., ZUKERMAN, B., AND HALPERIN, D. 2007. 2d arrangements. In *CGAL User and Reference Manual*, C. E. Board, Ed., 3.3 ed.
- WEIN, R., FOGEL, E., ZUKERMAN, B., AND HALPERIN, D. 2007. Advanced programming techniques applied to cgal's arrangement package. *Comput. Geom. Theory Appl.* 38, 1-2, 37–63.
- YAP, C. K. 1997. Towards exact geometric computation. *Comput. Geom. Theory Appl.* 7.