Self-Intersection Problems and Approximate Implicitization ⋆

Jan B. Thomassen[1,2]

[1] Centre of Mathematics for Applications, University of Oslo
URL: http://www.cma.uio.no
janbth@math.uio.no
[2] SINTEF Applied Mathematics
URL: http://www.sintef.no

Abstract. We discuss how approximate implicit representations of parametric curves and surfaces may be used in algorithms for finding self-intersections. We first recall how approximate implicitization can be formulated as a linear algebra problem, which may be solved by an SVD. We then sketch a self-intersection algorithm, and discuss two important problems we are faced with in implementing this algorithm: What algebraic degree to choose for the approximate implicit representation, and – for surfaces – how to find self-intersection *curves*, as opposed to just points.

1 Introduction

Self-intersection algorithms are important for many applications within CAD/ CAM. Suppose we try to create an offset surface within a CAD system with an offset distance larger than the radius of curvature at some points on the surface. Offset surfaces are not in general rational, and it is therefore necessary to make a NURBS approximation. In a typical CAD-system, surfaces are often represented in terms of degree $(3, 3)$ NURBS surfaces. A NURBS approximation of such an offset surface is shown in fig. 1. We would like to have parts of this surface trimmed away, or maybe even to have it rejected as an illegal surface. This requires an algorithm to find the self-intersection curves, or at least to detect that there are self-intersections.

Implicitization is the process of converting a parametric representation of a curve or surface into an implicit one. Figure 2 shows the implicit representation of a cubic curve with a node, given by the zero contour of the implicit function. Implicit representations have many uses. One example is finding intersections between different objects. Ray tracing is a simple case of this, where the parametric form of the ray – a straight line – is inserted into the implicit equation for a surface in order to get an equation in the parameter of the ray. This can then be solved to find the point, or points, of intersection. Note that self-intersections cannot be treated in this relatively simple way because inserting the parametric form into the implicit form for the same object gives identically zero. Thus, finding true self-intersections is a much harder problem.

⋆ This work was supported by the EU project GAIA II, IST-2001-35512

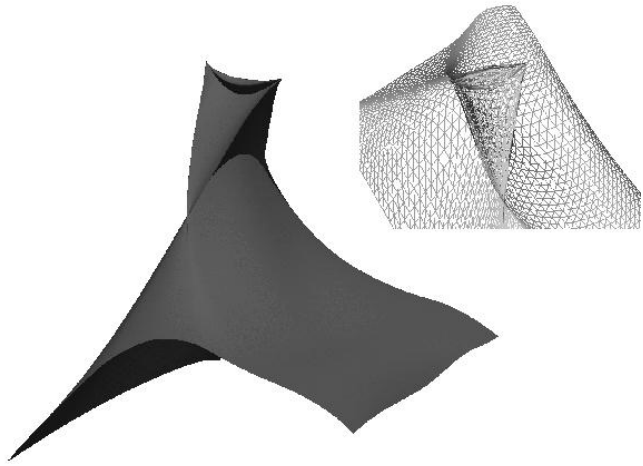

Fig. 1. NURBS approximation of an offset surface. Made within the CAD-system 'thinkdesign' from CAD-vendor think3.

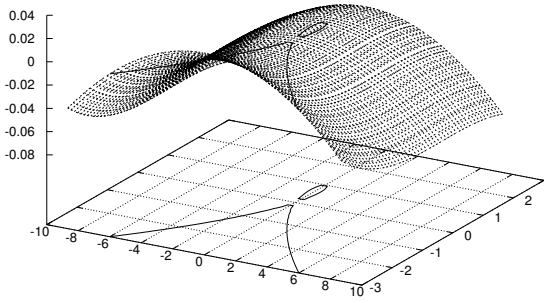

Fig. 2. Implicit representation of a nodal cubic

In this paper, we discuss an algorithm for finding self-intersections of NURBS curves and surfaces. This algorithm has in part grown out of the long-time experience collected at SINTEF [4, 12], and is still under development. The intersection problem has been studied a lot during the last few decades; for a partial list of references, see [12]. The present algorithm is optimized towards giving certifiable answers with respect to the self-intersections, rather than for speed. Therefore, central to the algorithm is an implicitization step and the use of implicit functions in detecting the self-intersections.

This will give us more information about the situation and secures the quality of the results compared to many brute force algorithms. Tangential (self-)intersections are in general difficult and will not be considered in this paper. We will therefore assume that all intersections are transversal.

Implicitization in CAGD has for a long time meant using classical algebraic techniques like resultants [10, 8] or Gröbner bases [3]. However, it is desirable to use approximate implicit representations of low algebraic degrees, which requires new methods. One recent method is the scattered data fitting method developed by Jüttler et al. [13]. This is very flexible but not accurate or fast. The method of choice for our algorithm is the one developed by Dokken [5, 6], which performs better on accuracy and speed [14]. This method essentially transforms the implicitization problem into the language of linear algebra. As far as we know, this is the simplest way to formulate the problem.

In the following, planar Bézier curves $\mathbf{p}(t)$ of degree n are defined in terms of $n+1$ control points $\mathbf{c}_i$ by

$$\mathbf{p}(t) = \sum_{i=0}^{n} \mathbf{c}_i B_{i,n}(t), \qquad t \in [0,1], \tag{1}$$

where the basis functions $B_{i,n}$ are Bernstein polynomials:

$$B_{i,n}(t) = \binom{n}{i} (1-t)^{n-i} t^i. \tag{2}$$

The unnormalized normal $\mathbf{n}$ is defined by

$$\mathbf{n}(t) \;=\; R_{\pi/2} \frac{d\mathbf{p}}{dt}(t), \tag{3}$$

where $R_{\pi/2} = \left(\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right)$ is a matrix that rotates a vector by an angle of $\frac{\pi}{2}$. Implicit curves of degree d are defined in terms of a polynomial function $q(\mathbf{x})$. In the actual implementation of the algorithm we use barycentric coordinates $\mathbf{x} = (u, v, w)$ with respect to a triangle enclosing the curve. This improves numerical stability. Thus, for the implicit function we have

$$q(\mathbf{x}) = \sum_{i+j+k=d} b_{ijk} B_{ijk,d}(\mathbf{x}), \qquad u+v+w = 1, \tag{4}$$

where

$$B_{ijk,d}(\mathbf{x}) = \frac{d!}{i!j!k!} u^i v^j w^k, \qquad i+j+k = d, \tag{5}$$

are the Bernstein polynomials over this triangle. These $\frac{1}{2}(d+1)(d+2)$ monomials constitute a basis for the polynomials of degree d. In barycentric coordinates q is a homogeneous polynomial.

We may now sketch the self-intersection algorithm for curves. We are given a NURBS curve of degree n in the plane and we want to find the parameter values that

correspond to self-intersections. Let us assume for simplicity that we are dealing with a non-rational[3] parametric curve – that is, the NURBS is in fact a B-spline.

1. Split the curve into Bézier segments.
2. For each segment $\mathbf{p}$, find candidates to self-intersection parameters. To do this,
   - find an implicit representation q,
   - form the quantity $\nabla q(\mathbf{p}(t)) \cdot \mathbf{n}(t)$ and find the roots.
3. For each pair of segments $\mathbf{p}_1$ and $\mathbf{p}_2$, find candidates to parameters for ordinary intersections. To do this,
   - find the implicit representations q_1 and q_2,
   - form the quantities $q_1(\mathbf{p}_2(t))$ and $q_2(\mathbf{p}_1(t))$ and find the roots.
4. Identify parameters for true self-intersections from the list of candidates. To do this, find pairs of parameters from this list whose corresponding 2D points match.

Although we have omitted the details, the structure of this algorithm should be straightforward. However, two things we will discuss further is the implicitization and the use of the quantity $\nabla q(\mathbf{p}(t)) \cdot \mathbf{n}(t)$. Thus, in the next section we will discuss exact implicitization. We will show that this is a linear algebra problem – in particular, it is an SVD problem. In sect. 3 we explain why $\nabla q(\mathbf{p}(t)) \cdot \mathbf{n}(t)$ is useful for finding self-intersections.

In sect. 4 we start considering surfaces, in particular tensor product B-splines. Unlike for curves, exact implicitization will typically require too high degrees, so if we want an algorithm similar to the one for curves, we need to find an approximate implicit representation. Approximate implicitization is the subject of sect. 5. Finally, in sect. 6 we discuss what are probably the two most important problems to solve before we can get an industrial strength surface intersection algorithm: What degree to choose for the approximate implicitization, and how to find self-intersection curves, as opposed to just points.

2 Implicitization

We are given a Bézier curve $\mathbf{p}(t)$ of degree n. We want to find a polynomial $q(\mathbf{x})$ such that the zero-set $\{\mathbf{x}|q(\mathbf{x}) = 0\}$ contains $\mathbf{p}$.

First of all, we want the degree d of q to be as small as possible for achieving exact implicitization. If $\mathbf{p}$ is sufficiently general, then $d = n$. If the true degree of $\mathbf{p}$ is smaller than n, for example if $\mathbf{p}$ is obtained by degree elevating a curve of lower degree, we still get an exact implicit representation, but with possible additional branches. This will typically produce phantom candidates for self-intersections in our algorithm, but that is not a problem because of the processing of the candidates at the end of the algorithm.

Thus we choose $d = n$, and we must find a nontrivial $\frac{1}{2}(d+1)(d+2)$-dimensional vector $\mathbf{b}$ of coefficients of q such that

$$q(\mathbf{p}(t)) = 0. \tag{6}$$

[3] In this paper, 'non-rational' means 'polynomial'. This is standard terminology in CAGD, see *e.g.* Farin's book [7].

This turns out to give a matrix equation in $\mathbf{b}$ [5, 6]. To see this we calculate $q(\mathbf{p}(t))$ by using the product rule for Bernstein basis functions:

$$B_{i,m}(t)B_{j,n}(t) = \frac{\binom{m}{i}\binom{n}{j}}{\binom{m+n}{i+j}} B_{i+j,m+n}(t). \tag{7}$$

From this we have that $q(\mathbf{p}(t))$ can be expressed in a Bernstein basis of degree nd. We organize the basis functions in an $(nd+1)$-dimensional vector $\mathbf{B}(t)^T = (B_{0,nd}(t), \ldots, B_{nd,nd}(t))$. Thus we find that inserting $\mathbf{p}$ in q yields the factorization

$$q(\mathbf{p}(t)) = \mathbf{B}(t)^T \mathbf{D}\mathbf{b}, \tag{8}$$

where $\mathbf{D}$ is an $M \times N$ matrix with $M = nd + 1$ and $N = \frac{1}{2}(d+1)(d+2)$. Since $\mathbf{B}$ is a basis, we see that the problem we need to solve is the matrix equation

$$\mathbf{D}\mathbf{b} = 0. \tag{9}$$

The standard method for solving a matrix equation like (9) is to use SVD [9]. The theory of the SVD states that we can find the decomposition $\mathbf{D} = \mathbf{U}\mathbf{W}\mathbf{V}^T$, where $\mathbf{U}$ is an $M \times N$ column-orthogonal, $\mathbf{W}$ an $N \times N$ diagonal, and $\mathbf{V}$ an $N \times N$ orthogonal matrix. The singular values of $\mathbf{D}$ are the numbers on the diagonal of $\mathbf{W}$. The solution to $\mathbf{D}\mathbf{b} = 0$ is an eigenvector corresponding to the vanishing singular values in $\mathbf{W}$. If there is exactly one singular value equal to zero, we can find the associated eigenvector from the corresponding column in $\mathbf{V}$. If there are more than one, any linear combination of the corresponding columns will solve (9). Note that we avoid problems with the trivial solution $\mathbf{b} = 0$ to (9). We may always normalize $\mathbf{b}$ and thereby $q(\mathbf{x})$.

An example of this method of implicitization is the nodal cubic Bézier curve in fig. 3. The result we have already seen in fig. 2. This implicitization is numerically very exact, and a plot of $q(\mathbf{p}(t))$ as a function of t would show zero to within machine precision.

Rational Bézier curves can be treated in essentially the same way. In barycentric coordinates a rational Bézier curve is given by

$$\mathbf{p}(t) = \frac{\sum_{i=0}^{n} w_i \mathbf{c}_i B_{i,n}(t)}{\sum_{i=0}^{n} w_i B_{i,n}(t)}, \tag{10}$$

where $\mathbf{c}_i$ are the control points and w_i are the weights. Since the implicit function q is homogeneous in barycentric coordinates, the denominator $\sum_i w_i B_{i,n}(t)$ leads to a polynomial that factors out in the expression for $q(\mathbf{p}(t))$. This means that the non-rational procedure can be applied with just the numerator $\sum_i w_i \mathbf{c}_i B_{i,n}(t)$ instead of $\mathbf{p}$ itself.

3 Finding Self-Intersections

The other issue we need to discuss is how to find the self-intersections of a Bézier curve $\mathbf{p}(t)$ once an implicit representation $q(\mathbf{x})$ is provided. For each self-intersection point we want to find the two corresponding parameters t_1 and t_2 such that

$$\mathbf{p}(t_1) = \mathbf{p}(t_2), \qquad t_1 \neq t_2. \tag{11}$$

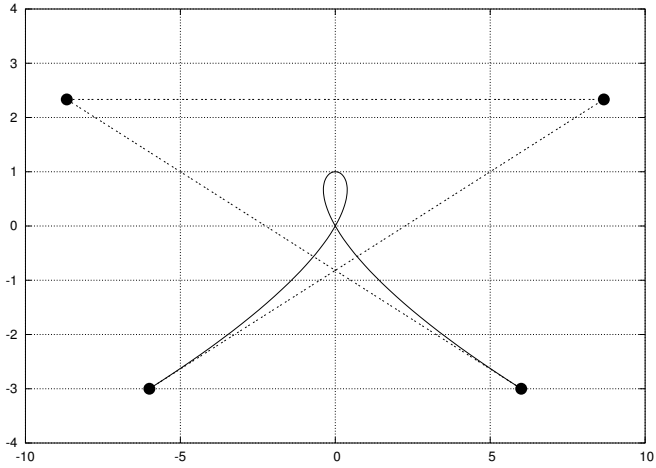

Fig. 3. Nodal cubic Bézier curve

This does not include cusps, which are "degenerate" self-intersections. We will briefly mention cusps later.

At a self-intersection point $\mathbf{p}_0 = \mathbf{p}(t_1) = \mathbf{p}(t_2)$ in the plane the gradient of the implicit function q vanishes: $\nabla q(\mathbf{p}_0) = 0$. In fact $\mathbf{p}_0$ is a saddle point of q, which looks like a hyperbolic paraboloid locally around $\mathbf{p}_0$. One way to find the self-intersection parameter values is to form $[\nabla q(\mathbf{p}(t))]^2$ and find the roots of this expression. However, this is not the optimal quantity for this purpose, because the polynomial degree is $2n(d-1)$, which is unnecessarily high, and all roots are located at minima. A better choice is the quantity

$$\nabla q(\mathbf{p}(t)) \cdot \mathbf{n}(t), \tag{12}$$

where $\mathbf{n}$ is the normal. The polynomial degree of (12) is $nd - 1$. This quantity is proportional to the projection of ∇q on the normal vector along the curve, and has the advantage that it changes sign at roots corresponding to transverse – i.e. non-tangential – self-intersections.

For the nodal cubic of fig. 3, the gradient dotted with the normal is plotted in fig. 4. The two roots occur at $t_1 = 0.25$ and $t_2 = 0.75$, which are the correct parameters for the self-intersection.

Roots of $\nabla q \cdot \mathbf{n}$ should be regarded as *candidates* for self-intersections. Indeed, ∇q may vanish along the curve $\mathbf{p}(t)$ if q is reducible so that the additional factors represent extra branches, or if $\mathbf{p}(t)$ reaches the endpoint, as we increase t, before the curve has time to complete the loop and self-intersect. This is why the last step of the algorithm is needed.

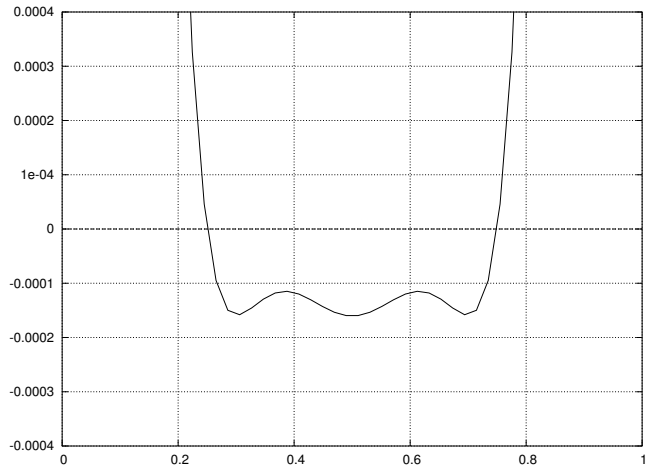

Fig. 4. $\nabla q \cdot \mathbf{n}$ for the nodal cubic of fig. 3

4 Surfaces

We now turn to surfaces. In this case we are given a tensor product NURBS surface. Let us assume this to be non-rational. Thus the surface consists of tensor product Bézier patches:

$$\mathbf{p}(u,v) = \sum_{i,j} \mathbf{c}_{ij} B_{i,n_1}(u) B_{j,n_2}(v), \tag{13}$$

where the degree is (n_1, n_2) and $\mathbf{c}_{ij}$ are the $(n_1+1)(n_2+1)$ control points. The unnormalized normal vector $\mathbf{n}(u,v)$ is defined by

$$\mathbf{n}(u,v) = \partial_u \mathbf{p}(u,v) \times \partial_v \mathbf{p}(u,v). \tag{14}$$

Like for curves we use barycentric coordinates $\mathbf{x} = (u,v,w,x)$ in our implementation, this time defined over a tetrahedron enclosing the surface. Then algebraic functions of degree d can be written

$$q(\mathbf{x}) = \sum_{i+j+k+l=d} b_{ijkl} B_{ijkl,d}(\mathbf{x}), \qquad u+v+w+x = 1, \tag{15}$$

with

$$B_{ijkl,d}(\mathbf{x}) = \frac{d!}{i!j!k!l!} u^i v^j w^k x^l, \qquad i+j+k+l = d, \tag{16}$$

the tetrahedral Bernstein basis. The polynomials $B_{ijkl,d}$ are now the $\frac{1}{6}(d+1)(d+2)(d+3)$-dimensional basis of degree d functions. Again, in barycentric coordinates $q(\mathbf{x})$ is a homogeneous function.

What we want now is an algorithm for finding self-intersections like the one for curves. *Ideally* this would look like:

1. Split the surface into Bézier patches.
2. For each patch $\mathbf{p}(u, v)$, find candidates to self-intersection curves in the (u, v)-plane. To do this,
   - find an implicit representation q,
   - form $\nabla q(\mathbf{p}(u, v)) \cdot \mathbf{n}(u, v)$ and find the roots.
3. For each pair of patches $\mathbf{p_1}(u, v)$ and $\mathbf{p_2}(u, v)$, find candidates to parameter curves for ordinary intersections. To do this,
   - find the implicit representations q_1 and q_2,
   - form $q_1(\mathbf{p}_2(u, v))$ and $q_2(\mathbf{p}_1(u, v))$ and find the zero curves in the (u, v)-plane.
4. Identify the curves in the parameter plane for true self-intersection curves from the list of candidates. To do this, find pairs of parameter curves from this list whose corresponding 3D curves match.

Unfortunately it is difficult to implement the algorithm in this way. There are essentially two problems. First, the necessary degree for exact implicitization is too high, both with respect to numerical stability and to speed. Hence the need to find an approximate implicit representation. For a chosen degree, the next section provides a review of how this can be done along the lines of the exact implicitization of sect. 2. However, this leads to the problem of what degree to choose. Second, self-intersection curves are difficult to handle.

In the current implementation of the algorithm, we have settled for a compromise to both these problems. We choose $d = 4$ for the approximate implicitization, based on the expectation that this degree is high enough to capture even complicated topologies. Our experience with this choice is good. Furthermore, instead of working with full self-intersection curves in the parameter plane, we work with points sampled on them. A sampling density must then be chosen, and a density of roughly 100×100 in the parameter plane for each patch gives acceptable results.

These two compromises lead to a change in the last step of the algorithm, since now also an iteration procedure is required for matching the points in 3D. This matching works roughly as follows. We loop over every pair of points from the list of candidates. Each pair of points in the (u, v)-plane corresponds to two points on the surface. We consider the squared Euclidean distance in 3D space between these two surface points. This is a four-variate function depending on the u and v parameter for the first and second points. Now we look for a minimum of this function. This is where the iterations come in – in our current implementation, a conjugate gradient method is used. If the iteration converges, and if the found minimum is a zero of the distance function, then we have a match and we record the corresponding parameters as points on a self-intersection curve. Then we go on with the next pair, etc. The points found in this way will in general not coincide exactly with the candidates we started with, but they still have the property that they lie on self-intersection curves.

An example is given by the offset surface we have already seen in fig. 1. As mentioned in the introduction, this is a realistic industrial example. It consists of many (rational) polynomial surface patches and has a very complex self-intersection structure. The result of running our algorithm on this surface is shown in fig. 5, where we have plotted points in the parameter plane that corresponds to points on the self-intersection curves in 3D space.

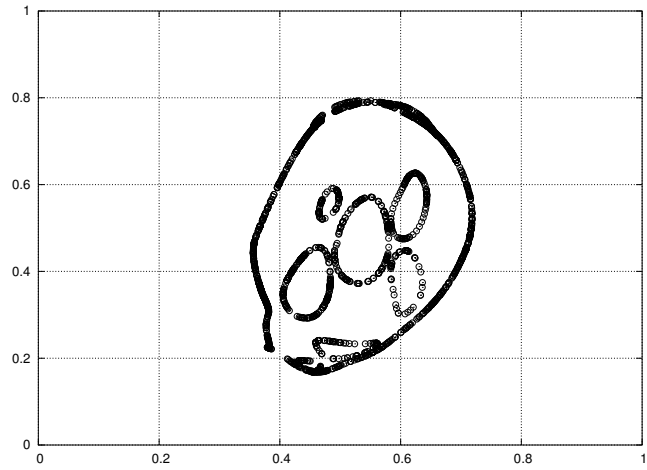

Fig. 5. Self-intersection points in the parameter plane of the offset surface in fig. 1

5 Approximate Implicitization

Although approximate implicitization is mostly relevant for surfaces, we will use curve language for simplicity in the following discussion. However, the results can easily be transferred to the surface case.

An implicitly defined curve $q(\mathbf{x}) = 0$ approximates a parametric curve $\mathbf{p}(t)$ within the tolerance ϵ if we can find a vector-valued function $\Delta\mathbf{p}(t)$ such that

$$q(\mathbf{p}(t) + \Delta\mathbf{p}(t)) = 0 \tag{17}$$

and

$$\max_t |\Delta\mathbf{p}(t)| \leq \epsilon. \tag{18}$$

$\Delta\mathbf{p}$ does not need to be continuous. In fact, for nontrivial topologies like self-intersections, $\mathbf{p} + \Delta\mathbf{p}$ may jump from one branch of q to another as we increase t.

We may Taylor expand (17) in $\Delta\mathbf{p}$ around $\mathbf{p}$:

$$q(\mathbf{p}(t)) + \nabla q(\mathbf{p}(t)) \cdot \Delta\mathbf{p} + \cdots = 0. \tag{19}$$

Thus, provided $\Delta\mathbf{p}$ is small and the coefficients of q are normalized in some way, the quantity $\max_t |q(\mathbf{p}(t))|$ is a meaningful measure of how well q approximates $\mathbf{p}$. This quantity is the "algebraic distance".

How do we find an approximate implicit function q such that the algebraic distance is as small as possible? Again we get an answer from linear algebra. If we insert the Bézier segment $\mathbf{p}(t)$ into a function q with unknown coefficients $\mathbf{b}$ we obtain the factorization in eq. (8),

$$q(\mathbf{p}(t)) = \mathbf{B}(t)^T \mathbf{D}\mathbf{b}. \tag{20}$$

Since $\mathbf{B}(t)$ is a Bernstein basis, $\|\mathbf{B}(t)\| \leq 1$ for all $t \in [0,1]$. Thus we get the inequality

$$\max_t |q(\mathbf{p}(t))| \leq \|\mathbf{D}\mathbf{b}\|. \tag{21}$$

Furthermore, the theory of the SVD tells us that

$$\min_{\|\mathbf{b}\|=1} \|\mathbf{D}\mathbf{b}\| = \sigma_{\min}, \tag{22}$$

where $\sigma_{\min}$ is the smallest singular value of $\mathbf{D}$. If we choose the corresponding eigenvector $\mathbf{b}_{\min}$ as the coefficients of q we have

$$\max_t |q(\mathbf{p}(t))| \leq \sigma_{\min}. \tag{23}$$

Thus, choosing the vector $\mathbf{b}_{\min}$ of coefficients gives us the implicit function we are looking for.

If we choose an algebraic degree d for q much lower than the required degree for exact implicitization, it may be that even the smallest singular value $\sigma_{\min}$ is greater than a given tolerance ϵ. It is then possible to improve the situation by subdividing. This leads us to consider convergence rates for approximate implicitization. Let us consider a curve $\mathbf{p}(t)$ and pick out an interval $[a,b] \subset [0,1]$ from the parameter domain with length h, that is, $b-a=h$. If we approximate the curve on this interval by an implicit function $q(\mathbf{x})$ of degree d we will have

$$|q(\mathbf{p}(t))| = O(h^{k+1}), \qquad t \in [a,b], \tag{24}$$

as h goes to zero, where the integer $k+1$ is the convergence rate. Dokken [5] proved that $k = \frac{1}{2}(d+1)(d+2)-2$. Likewise for a surface $\mathbf{p}(u,v)$ we get a convergence rate from

$$|q(\mathbf{p}(u,v))| = O(h^{k+1}) \tag{25}$$

with $k = \left\lfloor \frac{1}{6}\sqrt{9+132d+72d^2+12d^3} - \frac{3}{2} \right\rfloor$ (see [5]). The notation $\lfloor \cdots \rfloor$ means that '$\cdots$' is rounded downwards to the nearest integer. Thus, for surfaces an approximate implicitization of degree 4 will have convergence rate $O(h^7)$.

If we are not satisfied with the algebraic distance we get from implicitizing a surface, we may dramatically improve the situation with a few subdivisions of the parameter plane. It is therefore a useful strategy to include subdivision in a self-intersection algorithm that is based on implicitization.

6 Two Open Problems

Finally, in this section we address the two problems mentioned previously: What degree d should we choose for the implicit representation of a surface in our algorithm? And, also for surfaces, how do we describe and find self-intersection *curves*, as opposed to just points?

6.1 What Degree d Should we Choose?

For NURBS curves there is no problem with using the required degree for exact implicitization in our algorithm. That is, for a curve of degree n, we may use $d = n$. For all realistic test cases we have tried, this gives a fast and reliable algorithm.

For a NURBS surface of degree (m, n), the required degree for exact implicitization is in general $d = 2mn$. Thus the realistic case of degree $(3, 3)$ surfaces needs $d = 18$. A polynomial in 3D of this degree will in general have 1330 terms, which gives very poor precision and long evaluation times. Furthermore, the matrix $\mathbf{D}$ in the implicitization becomes so large that the SVD is very slow and has problems with converging.

There are several possibilities to overcome this. One is to choose some fixed low degree d and hardcode this in the algorithm. Another is to use some adaptive procedure where we start with a low value for d and then increase it until some requirement is met. We have experimented with both of these possibilities. The requirement in the adaptive procedure was that the smallest singular value in the SVD of $\mathbf{D}$ should be smaller than some number. However, it turned out to be very time consuming and not noticeably better than the hardcoding possibility. As already mentioned, $d = 4$ leads to acceptable results in all test cases we have tried.

Could we just as well manage with $d = 3$, which might lead to a faster algorithm? After all, implicitly defined surfaces of degree 3 are the simplest ones that may exhibit self-intersection curves. However, for $d = 3$ it turns out that the quality of the candidates to self-intersection points obtained from sampling the zeros of $\nabla q \cdot \mathbf{n}$ are very poor. This means that the iteration procedure in the matching step leads either to discarding most of the candidates or to produce a large number of redundant matches which is time consuming. On the other hand, for $d = 4$ iteration leads to matching for most of the candidates without too much redundancy.

An example is the surface shown in fig. 6, which has a closed self-intersection curve. The degree of the exact implicit representation is 6. Candidates and the result after iteration for $d = 3$ is shown in fig. 7, while the result for $d = 4$ is shown in 8.

Nevertheless, it is a suboptimal solution to hardcode the degree once and for all. It would be better to be able to determine for each surface patch which degree would be most suitable. But for the moment it is an open problem how to do that efficiently inside the algorithm.

6.2 How do we Find Self-Intersection *Curves* (as Opposed to *Points*)?

A possible strategy to finding self-intersection curves is to first find points on these curves and then use a marching technique to "march them out". The goal here would be to obtain a description of the intersection curves in terms of B-spline (or NURBS) curves in the parameter domain. We have already said a lot about finding such points, so let us discuss marching.

Marching is an iterative procedure, which can be used to trace out intersection curves [1, 2] (see [12] for more references). We start with a point on or nearby the curve, and then proceed by taking small steps at a time in a cleverly chosen direction until some stopping criterion is met. For instance, we stop when we reach the edge of the parameter domain or when the curve closes on itself.

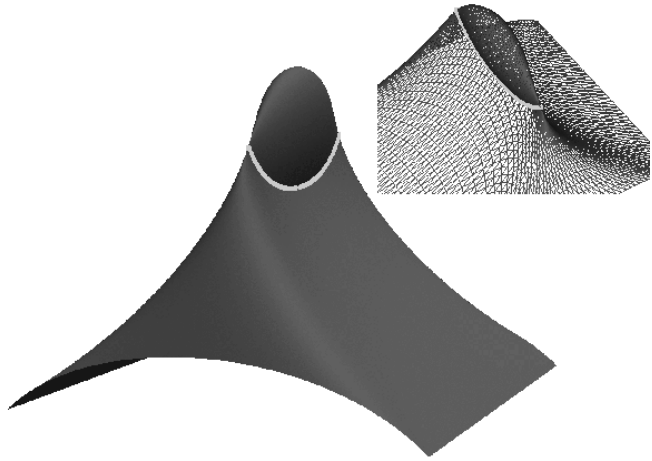

Fig. 6. Surface with points on self-intersection curve. Supplied by think3.

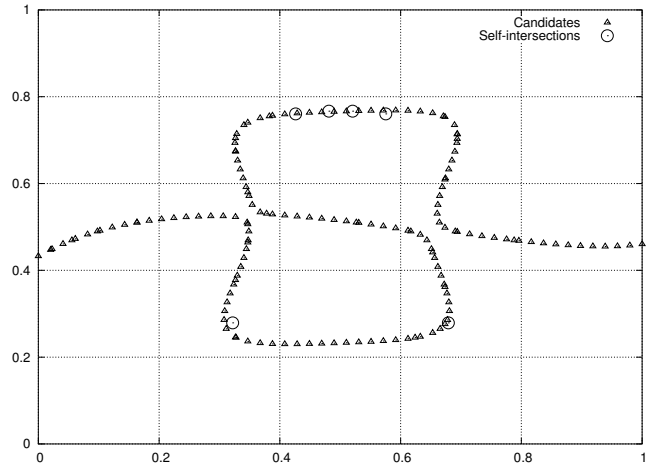

Fig. 7. Parameter plane with candidates and self-intersections obtained after iteration for the surface in fig. 6 for Degree 3

One problem we are faced with when marching out a self-intersection curve is the marching onto a singularity, which in this case means a point on the surface where the unnormalized normal vanishes. The normal is used in order to find the direction of the next step in the marching. A singularity of this kind is a cusp. For example, the surface in fig. 6 has two singularities, plotted in fig. 9. Closed self-intersection curves typically have singularities on them.

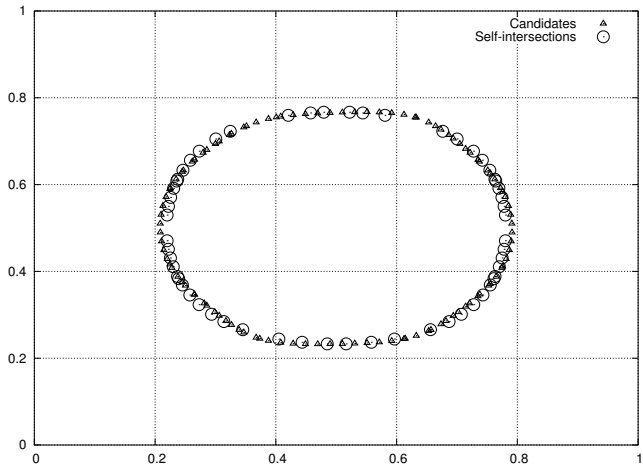

Fig. 8. Parameter plane with candidates and self-intersections obtained after iteration for the surface in fig. 6 for Degree 4

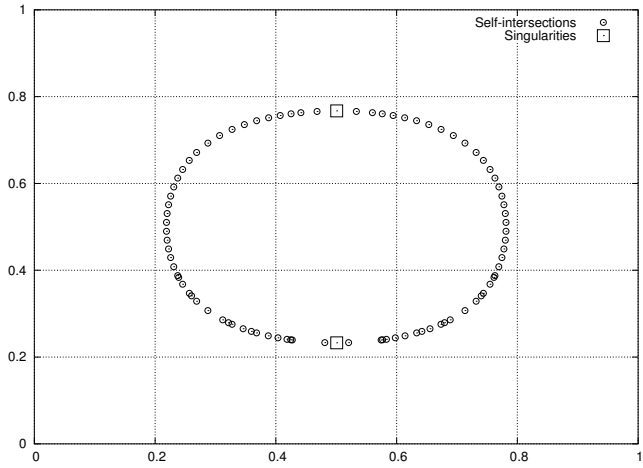

Fig. 9. The parameter plane of the surface in fig. 6, with self-intersection points and two cusp-like singularities

Thus, the solution is to identify all the isolated singularities and start the marching *from* these points. It is not difficult to find isolated cusp singularities. For instance, we may look for the zeros of $[\mathbf{n}(u, v)]^2$, which are also minima.

Unfortunately in real life things are not that easy. It may often happen that entire curves of cusp singularities exist. An example of this is the surface in fig. 10. Points sampled from the cusp curves and self-intersection curves in the parameter plane are plotted in fig. 11. As we may see from the plot, in order to start marching we would need

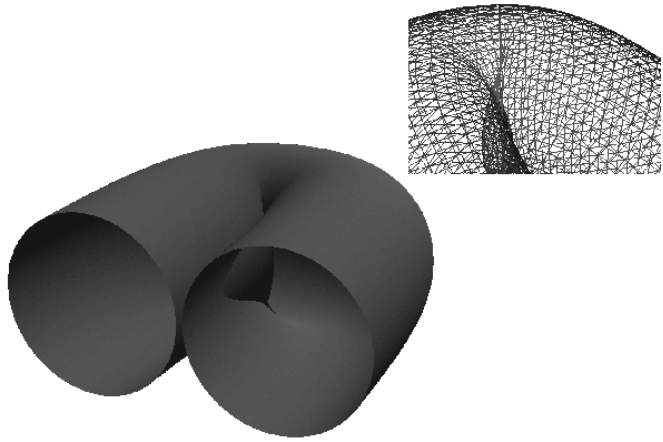

Fig. 10. Bent pipe surface with “supersingularities”. Surface provided by think3.

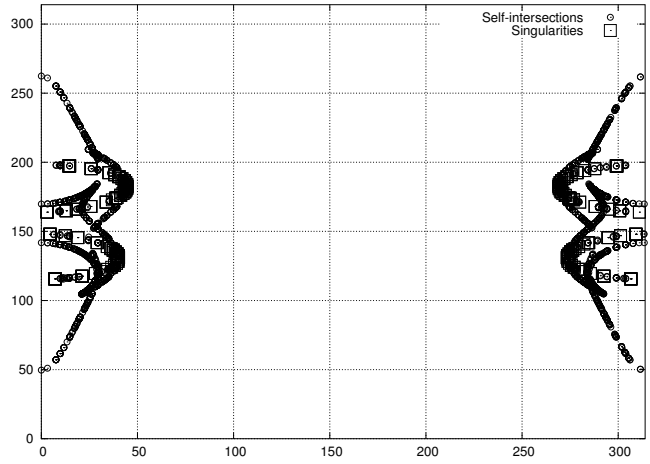

Fig. 11. Parameter plane of the pipe surface with self-intersection points and points on the cusp curves

to locate an intersection point between a self-intersection curve and a cusp curve. This is a very special kind of singularity, let us call it a “supersingularity”. At the moment we do not know how to properly characterize and find such supersingular points. Until we do, the treatment of self-intersection curves in such cases is another open problem.

7 Conclusion

We have described how we may use implicit representations of curves and surfaces in an algorithm for finding self-intersections. We have also described how to find the implicit representation given a NURBS curve or surface. We do this by formulating it as a linear algebra problem. For curves we may use an exact implicitization, while for surfaces it is necessary to use an approximate implicitization.

By using the implicit representation q and the normal vector $\mathbf{n}$ we are able to find candidates for the parameters of self-intersections by looking for the roots of $\nabla q \cdot \mathbf{n}$. For curves, where we use exact implicitization, the self-intersection parameters are included in the list of candidates. For surfaces, it is necessary to process the candidates with an iterative procedure to match them in order to get points on the self-intersection curves.

Surfaces are much more challenging than curves, and we have described two open problems: How do we choose the degree for the approximate implicitization, and how do we deal with the fact that self-intersections are in general curves?

Acknowledgements I thank Tor Dokken for discussions, and for reading and commenting on the manuscript.

References

1. C.L. Bajaj, C.M. Hoffmann, R.E. Lynch, and J.E.H. Hopcroft, Tracing surface intersections, *Computer Aided Geometric Design* 5 (1988) 285–307.
2. R.E. Barnhill, and S.N. Kersey, A marching method for parametric surface/surface intersection, *Computer Aided Geometric Design* 7 (1990) 257–280.
3. D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms*, Second Edition, Springer-Verlag, New York, 1997.
4. T. Dokken, V. Skytt, and A.-M. Ytrehus, Recursive Subdivision and Iteration in Intersections and Related Problems, in Mathematical Methods in Computer Aided Geometric Design, T. Lyche and L. Schumaker (eds.), Academic Press, 1989, 207–214.
5. T. Dokken, *Aspects of Intersection Algorithms and Applications*, Ph.D. thesis, University of Oslo, July 1997.
6. T. Dokken, and J.B. Thomassen, Overview of Approximate Implicitization, in *Topics in Algebraic Geometry and Geometric Modeling*, AMS Cont. Math. 334 (2003), 169–184.
7. G. Farin, *Curves and surfaces for CAGD: A practical guide*, Fourth Edition, Academic Press, 1997.
8. R.N. Goldman, T.W. Sederberg, and D.C. Anderson, Vector elimination: A technique for the implicitization, inversion, and intersection of planar rational polynomial curves, *Computer Aided Geometric Design* 1 (1984) 327–356.
9. W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes in C*, Second Edition, Cambridge University Press, 1992.
10. T.W. Sederberg, D.C. Anderson, and R.N. Goldman, Implicit Representation of Parametric Curves and Surfaces, *Comp. Vision, Graphics, and Image Processing* **28**, 72–84 (1984).
11. T.W. Sederberg, J. Zheng, K. Klimaszewski, and T. Dokken, Approximate Implicitization Using Monoid Curves and Surfaces, *Graph. Models and Image Proc.* **61**, 177–198 (1999).
12. V. Skytt, Challenges in surface-surface intersections, these proceedings.

13. E. Wurm, and B. Jüttler, Approximate implicitization via curve fitting, in L. Kobbelt, P. Schröder, H. Hoppe (eds.), *Symposium on Geometry Processing*, Eurographics, ACM Siggraph, New York 2003, 240–247.
14. E. Wurm, and J.B. Thomassen, Deliverable 3.2.1 – Benchmarking of the different methods for approximate implicitization, Internal report in the GAIA II project, October 2003.