Closest Points, Moving Surfaces, and Algebraic Geometry

Jan B. Thomassen, Pål H. Johansen, and Tor Dokken

Abstract. This paper presents a method for computing closest points to a given parametric surface based on "moving surfaces". Moving surfaces are implicitly defined objects that allow us to formulate the problem in terms of two *univariate* polynomial equations. The idea is to obtain a faster and more reliable method of computing closest points, compared to conventional methods based on Newton iterations. We also describe an implementation of our algorithm which – although not being fast – is very reliable.

§1. Introduction

In this paper, we present a new method for calculating closest points to a parametric surface. The method is based on algebraic techniques, in particular on moving surfaces. Moving surfaces are objects that have previously been used for implicitization [6], but the closest point problem now provides another application of these.

Recently, there has been renewed interest in exploring links between geometric modeling and algebraic geometry [5]. The work presented in this paper is a part of this trend, and extends work from the European Commission project GAIA II (see the Acknowledgements). Algebraic geometry has many uses in geometric modeling, including such applications as point classification, implicitization, intersection and self-intersection problems, ray-tracing, etc. It was therefore natural to ask whether algebraic geometry also can be used in algorithms for computing closest points.

The closest point problem is a generic problem in CAGD. Applications include surface smoothing, surface fitting, and curve or surface selection. The closest point problem can be described in the following way. We are given a parametric surface $\mathbf{p}(u, v)$ and a point $\mathbf{x}_0$ in space. We want to find the point $\mathbf{p}_{\mathrm{cl}}$ on the surface that is closest to $\mathbf{x}_0$, or more precisely, we want to find the parameters $(u_{\mathrm{cl}}, v_{\mathrm{cl}})$ of $\mathbf{p}_{\mathrm{cl}}$.

The conventional way to compute closest points involves iterative methods, like Newton's method, to minimize the distance function from $\mathbf{x}_0$ to a point on the surface. This leads to solving a set of two polynomial equations in u and v,

$$\begin{aligned} (\mathbf{x}_0 - \mathbf{p}(u,v)) \cdot \mathbf{p}_u(u,v) &= 0, \\ (\mathbf{x}_0 - \mathbf{p}(u,v)) \cdot \mathbf{p}_v(u,v) &= 0, \end{aligned} \tag{1}$$

for the footpoints to $\mathbf{x}_0$. We recall that a *footpoint* $\mathbf{p}$ to $\mathbf{x}_0$ is a point on the surface such that the vector $(\mathbf{x}_0 - \mathbf{p})$ is orthogonal to the tangent plane at $\mathbf{p}$. Eqs. (1) express an orthogonality condition: The vector $(\mathbf{x}_0 - \mathbf{p}_{\mathrm{cl}})$ is orthogonal to the tangent vectors $\mathbf{p}_u(u_{\mathrm{cl}}, v_{\mathrm{cl}})$ and $\mathbf{p}_v(u_{\mathrm{cl}}, v_{\mathrm{cl}})$ at the closest point. This is illustrated in Fig. 1.

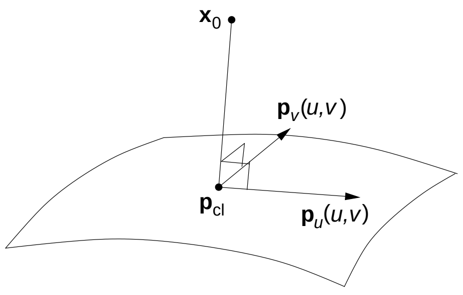

Fig. 1. Orthogonality conditions for the closest point.

One disadvantage of iterative methods is that we need an initial guess. It is a problem to come up with a good initial guess [3]. A bad guess may give a sequence of iterations that does not converge, or that converges to the wrong solution. Furthermore, if a large number of closest points needs to be computed, the method may be slow.

Another way to solve Eqs. (1) is to use subdivision techniques. An example of this is Bézier clipping [4]. These methods are often robust and effective, but may be unstable and use a long time to converge for some difficult surfaces, like surfaces with singularities. Such methods are probably the methods of choice in real applications, but we will not discuss them further here.

The method we propose in this paper for solving the closest point problem uses moving surfaces, as already mentioned. A moving surface in our setting is a one-parameter family of surfaces. We construct two such surfaces: one moving in the u-direction and one moving in the v-direction. The two moving surfaces give us two polynomial equations that are *univariate.* Univariate polynomial equations can be solved fast with a recursive solver and all roots may be found on the interval of interest within a predefined accuracy. This will give an algorithm that does not need any initial guess, has no convergence problems, and is fast when many closest points are to be calculated.

We may use elimination theory and Sylvester resultants to construct the moving surfaces in our method. From this construction, we obtain formulas for the algebraic degrees of the geometric objects involved when the surfaces addressed are Bézier surfaces.

We also describe an implementation of an algorithm for computing closest points based on the moving surface method. In this implementation, we construct the moving surfaces by solving a system of linear equations, rather than by using resultants. The implementation produced accurate results when run on test cases of biquadratic Bézier surfaces. Unfortunately, it couldn't be applied to bicubic surfaces due to memory shortage when building certain matrices necessary for the construction of the moving surfaces.

The organization of the paper is the following. In the following section, we describe the way we use moving surfaces and the idea behind our method. In Section 3, we analyze the method by using elimination theory and Sylvester's resultant, which gives us formulas for the algebraic degrees of the moving surfaces in the scheme. In Section 4, we present an algorithm for our method and describe some results we have obtained from implementing it. Finally, Section 5 is a discussion of these results.

§2. The Underlying Idea

Our method involves *moving surfaces*, which have been introduced by Sederberg for implicitization [6]. In that context, a moving surface is an implicit surface depending on two parameters, but in our setting a moving surface is a one-parameter family of implicit surfaces. Let us make the assumption that we are dealing with parametric surfaces that are single rational patches. Thus, a moving surface $q(\mathbf{x};u)$, depending on the parameter u, is given by

$$q(\mathbf{x};u) \quad = \quad \sum_{i=0}^{N} q_i(\mathbf{x}) B_{i,N}(u). \tag{2}$$

Here, $B_{i,N}(u)$ are Bernstein basis polynomials of degree N, and $q_i(\mathbf{x})$ is a set of $N+1$ algebraic functions. In other words, q is given in terms of a Bernstein polynomial in u of degree N, where the coefficients q_i are implicit surfaces. Furthermore, the moving surface q *follows* a surface $\mathbf{p}(u,v)$ (in the parameter u) if

$$q(\mathbf{p}(u,v);u) \quad = \quad 0. \tag{3}$$

Moving surfaces may in this way follow a parametric surface in either the u or the v direction.

How can we make use of such moving surfaces? Suppose we find a moving surface $q_1(\mathbf{x};u)$ with the following properties:

- q_1 follows the given surface $\mathbf{p}(u,v)$ in u. This means that the surface defined by $q_1(\mathbf{x};u) = 0$ intersects $\mathbf{p}$ in u-isoparameter curves.
- q_1 is orthogonal to $\mathbf{p}$ for each u.
- q_1 is ruled for each u. More precisely, it is swept out by lines spanned by the normal $\mathbf{n}(u,v)$ along the u-isoparameter curves.

Then, for a given point $\mathbf{x}_0$ in space, the equation

$$q_1(\mathbf{x}_0;u) \quad = \quad 0 \tag{4}$$

is a *univariate* equation for the u-parameter of all footpoints to $\mathbf{x}_0$. An example of a moving surface with these properties is shown in Figure 2. Clearly, we may have a similar moving surface q_2 in the v direction. In

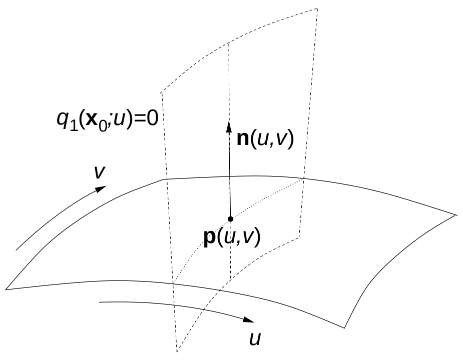

Fig. 2. A moving surface q_1 that intersects $\mathbf{p}$ at u-isocurves, is orthogonal to it, and is ruled.

the following, the subscript 1 or 2 on q refers to either u or v.

A possible exception to this situation is that we are dealing with certain non-generic surfaces, like surfaces of revolution. For these surfaces some points (like those lying on the axis of revolution) may give, not *footpoints*, but *footcurves*. I.e. the set of points with the same distance to $\mathbf{x}_0$ is a curve on the surface. This is presumably a problem for most methods of computing closest points, and requires a separate discussion. For simplicity we assume that all the surfaces we consider are sufficiently generic for this to happen.

Based on the considerations above, we propose a method for computing closest points in two steps:

1. Preprocessing. Construct two moving surfaces: q_1 for the u-direction, and q_2 for the v-direction. This is done once for each surface.

2. For each given point $\mathbf{x}_0$, use the two moving surfaces to get two univariate equations in u and v:

$$\begin{aligned} q_1(\mathbf{x}_0; u) &= 0, \\ q_2(\mathbf{x}_0; v) &= 0. \end{aligned} \tag{5}$$

   Check each pair of solutions (u, v) to these equations, along with the closest point on the border, to find which one corresponds to the closest point.

Finding the closest point on the border amounts to running a similar algorithm for the four border curves.

A sketch of a situation where we get a solution u_0 and v_0 from Step 2 is shown in Figure 3.

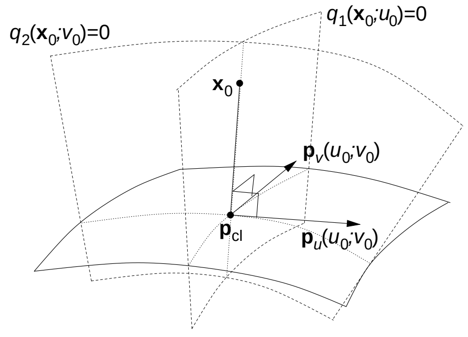

Fig. 3. When the solutions u_0 and v_0 are found in Step 2, we can draw the moving surfaces for these two parameter values. The point $\mathbf{x}_0$, the closest point $\mathbf{p}_{\mathrm{cl}}$, and the straight line between them, lie on both of these surfaces.

Let us also make a remark about curves. A similar construction works for curves, both in 2D and 3D. In 2D we have moving lines, while in 3D we have moving planes. Since lines and planes are described implicitly by algebraic functions that are linear, the algorithms become simpler. Furthermore, for curves there is only one equation in Step 2. This equation is in fact *equivalent* to the orthogonality condition $(\mathbf{x}_0 - \mathbf{p}(t)) \cdot \mathbf{p}'(t) = 0$.

§3. Degrees of the Moving Surfaces

The two moving surfaces described in the previous section can be analyzed more formally. In this section we will use elimination theory, in particular Sylvester's resultant, to perform this analysis [1]. We will assume that the surface $\mathbf{p}$ is a single polynomial patch, i.e. a Bézier patch.

In this case we obtain formulas for the algebraic degrees involved in q_1 and q_2 given a parametric surface of bidegree (n_u, n_v). Referring back to the form (2) of a moving surface, the required degrees are:

$$\begin{array}{rcccl} d_1 & \equiv & \deg_{\mathbf{x}}(q_1) & = & \text{the degree of } q_1 \text{ (or } q_{1,i}) \text{ in } \mathbf{x} \\ d_2 & \equiv & \deg_{\mathbf{x}}(q_2) & = & \text{the degree of } q_2 \text{ (or } q_{2,i}) \text{ in } \mathbf{x} \\ N & \equiv & \deg_u(q_1) & = & \text{the degree of } q_1 \text{ (or } B_{i,N}) \text{ in } u \end{array}$$

It turns out that $\deg_u(q_1)$ is equal to $\deg_v(q_2)$ so we need only one N. This is connected with the fact that N counts the number of possible footpoints, and this is given by the number of roots of q_1 and q_2, respectively.

Thus we have a parameterized surface $\mathbf{p} : \mathbb{R}^2 \to \mathbb{R}^3$, where $\mathbf{p}$ is given by three polynomials $p_1, p_2, p_3 \in \mathbb{R}[u, v]$ of degree (n_u, n_v). We assume that this description of the surface is sufficiently general, so that the degrees cannot be reduced. Now let V be the set of points $(u, v, \mathbf{x}) \in \mathbb{R} \times \mathbb{R} \times \mathbb{R}^3$ such that $\mathbf{x}$ is on the normal of $\mathbf{p}$ given by the parameter values (u, v).

The set V is described by the two equations

$$\begin{aligned} F_1(u, v, \mathbf{x}) := (\mathbf{x} - \mathbf{p}) \cdot \mathbf{p}_u &= 0, \\ F_2(u, v, \mathbf{x}) := (\mathbf{x} - \mathbf{p}) \cdot \mathbf{p}_v &= 0. \end{aligned} \tag{6}$$

The points satisfying these equations make a variety in $\mathbb{R}^5$.

Using a resultant, we can eliminate one variable, and get one polynomial defining a hypersurface in $\mathbb{R}^4$. If we eliminate u, this set of points is exactly the set $V' = \{(v, \mathbf{x}) \in \mathbb{R} \times \mathbb{R}^3 \mid \exists u \in \mathbb{R} \text{ s.t. } F_1(u, v, \mathbf{x}) = F_2(u, v, \mathbf{x}) = 0\}$, which corresponds to the moving surface q_2.

We want to determine the degrees in v and $\mathbf{x}$ of the equation defining V'. First, we write

$$\begin{aligned} F_1(u, v, \mathbf{x}) &= \sum_{i=0}^{2n_u-1} f_i(v, \mathbf{x}) u^i, \\ F_2(u, v, \mathbf{x}) &= \sum_{i=0}^{2n_u} g_i(v, \mathbf{x}) u^i, \end{aligned} \tag{7}$$

and then use the Sylvester resultant to eliminate u. By examining the Sylvester matrix, we can determine the degrees of the equation defining V'. The Sylvester matrix is a square matrix of size $(4n_u - 1) \times (4n_u - 1)$.

It looks like this:

$$\left(\begin{array}{cccc|ccc}
f_0 & 0 & \cdots & 0 & g_0 & \cdots & 0 \\
f_1 & f_0 & \cdots & 0 & g_1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
f_{2n_u-1} & f_{2n_u-2} & \cdots & f_0 & g_{2n_u-1} & \cdots & g_1 \\
0 & f_{2n_u-1} & \cdots & f_1 & g_{2n_u} & \cdots & g_2 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & f_{2n_u-1} & 0 & \cdots & g_{2n_u}
\end{array}\right)$$

There are $2n_u$ columns to the left, and the degree in v is $2n_v$ for each of these entries. There are $2n_u - 1$ columns to the right, each entry being of degree $2n_v - 1$. The total degree in v of the resultant is thus

$$N \quad = \quad 4n_u n_v + (2n_u - 1)(2n_v - 1). \tag{8}$$

Note the symmetry of this expression with respect to n_u and n_v, which confirms what we said previously about needing only one N.

The degree in $\mathbf{x}$, that is, d_2, is a little trickier to work out. The polynomials $f_0, \ldots, f_{n_u-1}$ are of degree 1 in $\mathbf{x}$, but the polynomials $f_{n_u}, \ldots, f_{2n_u-1}$ are of degree 0. Furthermore, the polynomials $g_0, \ldots, g_{n_u}$ are of degree 1 and the polynomials $g_{n_u+1}, \ldots, g_{2n_u}$ are of degree 0 in $\mathbf{x}$. This means that the bottom n_u rows are of degree 0 in $\mathbf{x}$ and the rest of the $3n_u - 1$ rows are of degree 1. The total degrees are therefore

$$d_1 \quad = \quad 3n_v - 1, \tag{9}$$
$$d_2 \quad = \quad 3n_u - 1. \tag{10}$$

As mentioned, the degree formulas for $d_{1,2}$ and N are derived for general parametrized surfaces, and as such are upper bounds. For some surfaces the degrees could be effectively lower. This happens, for example, if the degree of $\mathbf{p}$ is artificially high, so it can be obtained from a degree elevation of a lower-degree parametrization. The degree can drop in other cases, but if the degree in v drops, then the corresponding degree in u will typically drop in the same way. For this reason, there will still be only one N.

A similar analysis can be carried out for rational surface patches. The degree formulas are then:

$$\begin{aligned}
N &= 9n_u n_v + (3n_u - 2)(3n_v - 2) \\
d_1 &= 4n_v - 2 \\
d_2 &= 4n_u - 2
\end{aligned} \tag{11}$$

Examples of the degrees for Bézier and rational surfaces of degrees (n, n) with n ranging from 1 to 4 is shown in Table 1. As far as we know,

	Bézier				Rational			
	$(1,1)$	$(2,2)$	$(3,3)$	$(4,4)$	$(1,1)$	$(2,2)$	$(3,3)$	$(4,4)$
$d_{1,2}$	2	5	8	11	2	6	10	14
N	5	26	61	113	10	52	130	244

Tab. 1. Degrees for Bézier and rational surfaces of degrees of the form (n, n). Since $n_u = n_v$ we also have $d_1 = d_2$.

these results are new[1]. The numbers d_1 and d_2 are the degrees of an algebraic surface that is perpendicular to a parametric surface along an entire isocurve, and this has not been noted before.

§4. Implementation of a Test Algorithm

To test our ideas, we have implemented an algorithm for computing closest points for tensor product Bézier surfaces. We have chosen not to use resultants for this. Instead we rely on solving a system of linear equations, which will be explained below. The reason is that this is a numerically very stable method, which allows us to use the Bernstein form for all polynomials in a straightforward way, which would not have been the case for resultant based methods. Besides, we do not get into possible problems with base points.

A central object in our implementation is the "moving ruled surface"

$$\mathbf{r}(u,v,w) \quad = \quad \mathbf{p}(u,v) + w\mathbf{n}(u,v), \tag{12}$$

where $\mathbf{p}$ is the given surface, $\mathbf{n}$ is the normal vector, and w is an additional parameter. This can be thought of as a *trivariate* tensor product Bézier object. For fixed (u_0, v_0), the line $\mathbf{r}(u_0, v_0, w)$, $w \in \mathbb{R}$, is orthogonal to the surface at $\mathbf{p}(u_0, v_0)$. In other words, all points on this line has $\mathbf{p}(u_0, v_0)$ as a footpoint.

Another property we have used in our implementation, is that evaluating an algebraic function $q(\mathbf{x})$ on an n-variate Bernstein tensor polynomial $\mathbf{r}(u_1, \ldots, u_n)$ yields a new n-variate Bernstein tensor polynomial. If we write $q(\mathbf{x}) = \sum_j b_j x^j$, where j is a multi-index, x^j is a monomial in (x, y, z) in multi-index form, and b_j are the coefficients, we have a factorization

$$q(\mathbf{r}(u_1, \ldots, u_n)) \quad = \quad \mathbf{b}^T \mathbf{D}^T \mathbf{B}(u_1, \ldots, u_n). \tag{13}$$

Here, $\mathbf{b}$ is the coefficients b_j organized in a vector, $\mathbf{D}$ is a matrix of numbers, and $\mathbf{B}(u_1, \ldots, u_n)$ is a basis of n-variate Bernstein tensor polynomials, also organized in a vector. If q is a degree d algebraic function

[1]We thank the referee for urging us to make this point.

and $\mathbf{r}$ is a degree $(m_1, \ldots, m_n)$ Bernstein polynomial, then $q(\mathbf{r})$ is a degree $(dm_1, \ldots, dm_n)$ Bernstein polynomial. In our implementation, we use evaluation routines for algebraic functions on Bernstein polynomials in order to find such matrix factorizations.

The moving surfaces q_1 and q_2 are defined by an array of coefficients. For $q_1(\mathbf{x}; u)$ we need to determine the coefficients $b_{1,i;j}$ of $q_{1,i}(\mathbf{x}) = \sum_j b_{1,i;j} x^j$, see Eq. (2). This means that we can use numerical linear algebra to find the vector $\mathbf{b}_1$ of coefficients in q_1 More precisely, we need to find a vector in the null-space of $\mathbf{D}_1$. We used a technique based on Gauss elimination and back-substitution for this, which is faster than, say, SVD of $\mathbf{D}_1$. This way of using numerical linear algebra has previously been used in implicitization, see [2].

The algorithm follows the two-step structure described in Section 2.

Step 1. Preprocessing

Input: A parametric surface $\mathbf{p}(u, v)$.

1. Construct a "moving ruled surface" $\mathbf{r}(u, v, w)$
2. Insert $\mathbf{r}$ into q_1 to get the equation $q_1(\mathbf{r}(u, v, w), u)) = 0$. This can be factored into the linear equation

$$\mathbf{B}^T(u, v, w)\mathbf{D}_1\mathbf{b}_1 \quad = \quad 0, \tag{14}$$

   where $\mathbf{b}_1$ is the vector of coefficients in $q_{1,i}$. Similarly for the v-direction.
3. Solve the matrix equation $\mathbf{D}_1\mathbf{b}_1 = 0$ by e.g. Gauss elimination and back-substitution. Similarly for the v-direction.

Output: The vectors $\mathbf{b}_1$ and $\mathbf{b}_2$, or equivalently, the moving surfaces q_1 and q_2.

Step 2. For each given point $\mathbf{x}_0$

Input: A point $\mathbf{x}_0$ in space.

1. Find the closest point on the boundary curves.
2. Insert $\mathbf{x}_0$ in q_1 and q_2 to get univariate equations in u and v:

$$\begin{aligned} q_1(\mathbf{x}_0; u) &= 0, \qquad (15) \\ q_2(\mathbf{x}_0; v) &= 0. \end{aligned}$$

3. Find all roots u_i and v_j.
4. Check each pair (u_i, v_j) and the closest point on the boundary to find the closest point.

Output: The parameters $(u_{\text{cl}}, v_{\text{cl}})$ of the closest point $\mathbf{p}_{\text{cl}}$.

	Moving surfaces		Newton's method
Average no. of hits	413		374
Running times	Full algo.	$\sim 1 - 2$ min	~ 5 s
	Just Step 2	~ 1 s	
Accuracy	$\sim 10^{-7}$		$\sim 10^{-13}$

Tab. 2. Average results for running the two closest point algorithms on ten random biquadratic surfaces. For details, see the text.

As an example, we tested the algorithm on a set of ten random biquadratic Bézier surfaces. That is, the control points were random points in the unit cube. We expect that within the family of biquadratic surfaces such surfaces will be a challenge for any closest point algorithm. For each surface, 1000 points were generated randomly in the bounding box, and their closest points on the surface were computed. However, points whose closest points were found to lie on the boundary were discarded. For comparison, we also implemented an algorithm based on Newton's method. We used a PC with two Intel Pentium 4 2.8GHz processors to run these algorithms. The results are shown in Table 2.

Table 2 reports the average number of closest points found (hits) for each surface. For both algorithms, less than half of the 1000 random points gave hits because a majority of the closest points were on the boundaries of the surfaces. (The surfaces had complicated geometries with lots of self-intersections.) The moving surfaces algorithm was consistently better for getting hits – Newton's method produced a lot of messages for "No convergence".

Running times were considerably longer for the full moving surfaces algorithm, with $1 - 2$ minutes. Most of this time is spent in the preprocessing step where the moving surfaces are constructed. When only Step 2 of the algorithm is considered it is much faster.

Finally, the accuracy given in Table 2 is an average of the errors for the reported closest points. The averaging is over the order of magnitude of the errors, i.e. it is an average of the log of the errors for each point. An error for a single point was defined in terms of the angle θ between the vector $(\mathbf{x}_0 - \mathbf{p}_{\mathrm{cl}})$ and the normal $\mathbf{n}(u_{\mathrm{cl}}, v_{\mathrm{cl}})$ at the computed closest point, see Figure 4. As we can see, Newton's method produced much better accuracy than the moving surfaces. Sources of error for the moving surfaces method are the building of the matrix $\mathbf{D}$, the Gauss elimination, the insertion of $\mathbf{x}_0$ to get the polynomial equations, and the solving of these equations. The results in Table 2, however, does not include iterative refinements.

Looking into the details for each surface – not shown in Table 2 – it turns out that there is a complementary property for the two algorithms:

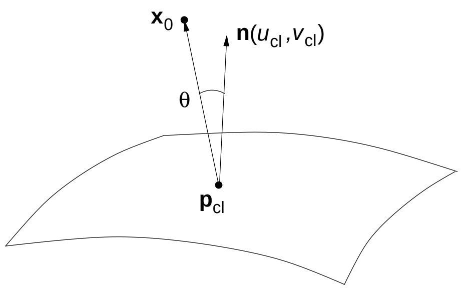
$\mathbf{x}_0$
$\mathbf{n}(u_{\mathrm{cl}},v_{\mathrm{cl}})$
θ
$\mathbf{p}_{\mathrm{cl}}$

Fig. 4. The error can be measured by the angle θ between the normal $\mathbf{n}$ at $\mathbf{p}_{\mathrm{cl}}$ and the vector to the point $\mathbf{x}$.

Surfaces that had a low accuracy for moving surfaces also had a high number of hits. For example, one random surface had an accuracy of 10^{-5} vs. 10^{-13} for moving surfaces and Newton's method, respectively, while the hit numbers were 265 vs. 193.

It is necessary to make some remarks about problems with the memory usage of our implementation of the moving surfaces algorithm. The amount of memory needed for the matrix $\mathbf{D}_1$ (or $\mathbf{D}_2$) was about 350 Mbytes for the biquadratic surface. This is a lot, but does not cause any problems. For a bicubic Bézier surface, however, the corresponding memory requirement is about 4 Gbytes with double precision! But even going to single precision was too much to handle for our PCs.

§5. Discussion

Moving surfaces provides a new method for computing closest points to a parametric surface. It is an alternative to the conventional algorithms based on iterations and Newton's method, or to subdivision methods.

Compared to a Newton based closest point algorithm, it takes a long time to set up the system of moving surfaces for each parametric surface, but a short time to compute the closest point once a point in space is given. This suggests that the potential use of the moving surfaces method is for situations where a large number of closest points are to be computed for each given surface, and where long preprocessing times are acceptable.

Furthermore, the moving surfaces method is better than Newton's method for actually finding the closest points for surfaces with complex geometry. Thus, if this kind of stability is desired, the moving surfaces method may also be a better choice, combined with an iterative refinement of the resulting closest points. In other words, the closest points found from the moving surfaces method could be used as the starting point of the Newton iterations.

However, we had problems with the implementation of the algorithm, due to memory shortage, when applied on the realistic case of bicubic surfaces. The large amount of memory is mainly used for building the matrices $\mathbf{D}_1$ and $\mathbf{D}_2$. In contrast, the amount of memory needed for storing the moving surfaces themselves corresponds to only $(N+1)(d_{1,2}+1)(d_{1,2}+2)(d_{1,2}+3)/6$ doubles (i.e. the dimensions of the vectors $\mathbf{b}_1$ and $\mathbf{b}_2$, respectively. In the bicubic case this number is 10230. This shows that we must find another way of implementing the algorithm, or that we must find a way to use moving surfaces together with approximations. But if we can afford the preprocessing of the coefficients of the moving surfaces we can get fast and accurate calculations of closest points.

Acknowledgments This work was supported by the European Commission project GAIA II, IST-2001-35512

§6. References

1. Cox, D., J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms*, Second Edition, Springer-Verlag, New York, 1996.
2. Dokken, Tor, and J. B. Thomassen, Overview of Approximate Implicitization, in [5].
3. Ma, Y. L., and W. T. Hewitt, Point Inversion and Projection for NURBS Curve and Surface: Control Polygon Approach, Comput. Aided Geom. Design **20** (2003), 79–99.
4. Nishita, T., T. W. Sederberg, and M. Kakimoto, Ray Tracing Trimmed Rational Surface Patches, Computer Graphics **24**, 1990, 337–345.
5. *Topics in Algebraic Geometry and Geometric Modeling*, proceedings of a workshop in Vilnius, Lithuania 2002, R. Goldman and R. Krasauskas (eds.), Contemporary Mathematics, Vol. **334**, 2003.
6. Sederberg, T. W., and F. Chen, Implicitization using Moving Curves and Surfaces, in Computer Graphics (SIGGRAPH 95 Conference Proceedings), ed. R. Cook, Vol. **29**, pp. 301–308, Addison–Wesley 1995.

Jan B. Thomassen and Pål H. Johansen
Centre of Mathematics for Applications
P.O. Box 1053 Blindern
NO–0316 Oslo
NORWAY
`jan.b.thomassen@cma.uio.no`
`hermunn@math.uio.no`
`http://www.cma.uio.no`

Tor Dokken
SINTEF ICT
P.O. Box 1024 Blindern
NO–0314 Oslo
NORWAY
`Tor.Dokken@sintef.no`
`http://www.math.sintef.no`