Efficient Collision Detection for Curved Solid Objects

Elmar Schömer
MPI für Informatik
Saarbrücken, Germany
schoemer@mpi-sb.mpg.de

Joachim Reichel
MPI für Informatik
Saarbrücken, Germany
reichel@mpi-sb.mpg.de

Thomas Warken
MPI für Informatik
Saarbrücken, Germany
warken@mpi-sb.mpg.de

ABSTRACT

The design-for-assembly technique requires realistic physically based simulation algorithms and in particular efficient geometric collision detection routines. Instead of approximating mechanical parts by large polygonal models, we work with the much smaller original CAD-data directly, thus avoiding precision and tolerance problems. We present a generic algorithm, which can decide whether two solids intersect or not. We identify classes of objects for which this algorithm can be efficiently specialized, and describe in detail how this specialization is done. These classes are objects that are bounded by quadric surface patches and conic arcs, objects that are bounded by natural quadric patches, torus patches, line segments and circular arcs, and objects that are bounded by quadric surface patches, segments of quadric intersection curves and segments of cubic spline curves. We show that all necessary geometric predicates can be evaluated by finding the roots of univariate polynomials of degree at most 4 for the first two classes, and at most 8 for the third class. In order to speed up the intersection tests we use bounding volume hierarchies. With the help of numerical optimization techniques we succeed in calculating smallest enclosing spheres and bounding boxes for a given set of surface patches fulfilling the properties mentioned above.

Categories and Subject Descriptors

I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Geometric Algorithms, Languages, and Systems*; I.6.m [**Simulation and Modeling**]: Miscellaneous

General Terms

Algorithms

Keywords

Manufacturing and Assembly Planning, Collision Detection, Computational Geometry, Geometric Interrogations and Reasoning

1. INTRODUCTION

Today, virtual reality techniques are commonly used in different areas. These are, for instance, the fields of virtual prototyping and assembly planning, where complex mechanical systems are designed and their assembly and functionality are simulated. An important task in the field of virtual reality is static collision detection, i.e., to decide whether or not two objects intersect. This problem is different from computing the intersection between two objects in CAD applications, where the complete topology of the intersection object has to be computed. An algorithm for static collision detection has only to answer the question if there is an intersection, and if so it must compute at least one point that witnesses the intersection. Moreover, not only the numerical robustness of the routines is important, but also their usability in real-time applications. This problem has been well studied in the past for polyhedral objects (see [11]), whereas there was only little effort spent on objects with curved surfaces. Curved objects were instead approximated by polyhedra than used directly. One reason for this is that the exclusive use of polyhedra was not seen as a strong restriction because any curved object can be approximated arbitrarily well by a polyhedron. Another reason is that the basic algorithms, e.g. intersection tests, can be implemented very easily, robustly and efficiently for polygons. But there are disadvantages of this approach, too. Obviously there is a tradeoff between the efficiency of the algorithms that operate on the objects, and the accuracy of the approximation. The running time of a collision detection algorithm, for example, depends intensely on the complexity of the representation of the objects. On the other hand, there are applications whose correctness also depends on the accuracy of the approximation. So it is natural to ask whether it is possible to handle objects with curved surfaces directly instead of approximating them. It seems to be worth spending more effort on the basic algorithms if one can reduce the complexity of the representation of the objects drastically.

Algorithm 1 FACETEST(f_1, f_2)

```
1: for all edges e of f1 do
2:    if e intersects f2 then
3:       return INTERSECTION
4: for all edges e of f2 do
5:    if e intersects f1 then
6:       return INTERSECTION
7: for all loops l of F1 ∩ F2 do
8:    determine a point p on l
9:    if (f1 contains p) and (f2 contains p) then
10:      return INTERSECTION
11: return NO_INTERSECTION
```

In this paper we present an efficient static collision test that operates on curved solids. The problem will be reduced to finding the roots of polynomials in one variable. We will identify classes of objects for which closed-form solutions exist. This means that all occurring polynomials have degree at most four. Experiments showed, that using Cardano's and Ferrari's formulae leads to sufficiently accurate results. Moreover, numerical methods are at least by a factor of ten slower. In a more general setting, we consider surfaces that possess an implicit representation of low degree (quadrics) and whose spatial trimming curves have a simple parametric form (quadric intersection curves and cubic spline curves). We show that all necessary geometric predicates can be evaluated by finding the roots of univariate polynomials of degree at most 8. Although NURBS surfaces are often used in CAD applications, they are not included in our object classes. The reason for this is that intersection tests involving NURBS (e.g. [3], [8]) are too time consuming for real-time applications at present. One fundamental problem in that context is loop detection (cf. [9]), which is similar to the surface-surface intersection-test that arises in our algorithm. For applications such as the above mentioned assembly planning, the restriction to (natural) quadrics and the torus as surface types is not a problem in general. The parts that are relevant for assembly processes are typically simple objects that are composed of natural quadrics and the torus. Therefore, it is worth developing algorithms that handle these surface types efficiently.

In order to speed up the collision test we use a hierarchy of bounding volumes, which allows one to quickly exclude a large number of object parts from consideration. In this way we reduce the number of calls to the basic routines, which consume most of the time during the computation. We present an algorithm that uses numerical optimization techniques to compute the smallest enclosing sphere for a given set of curved faces. Moreover we present an algorithm to compute an oriented bounding box with locally minimal volume.

2. A GENERIC ALGORITHM

In this section we describe a generic algorithm for the intersection test between two solids that are given in boundary representation. More precisely, we want to test whether the boundaries of the solids intersect, and if this is not the case, we want to decide whether one object is contained in the other one. If the intersection test for the boundaries determines that there is no intersection, then we know that the objects are either disjoint or one of them is contained in the other one. In order to decide which case applies, we choose a point on the first solid and test whether it is inside or outside the other one, and vice versa. This point-in-solid-test can be done by sending a ray from the query point to infinity and counting the number of intersections with the solid. If this number is even then the point is outside, otherwise it is inside.

In order to check whether the boundaries of two solids intersect we proceed as follows. For each face f_1 of solid 1 and each face f_2 of solid 2 we check whether f_1 and f_2 intersect. If we find a pair of intersecting faces, then the solids intersect, otherwise they do not. Instead of testing all possible pairs of faces, we can use a hierarchy of bounding volumes to exclude large parts of the objects from consideration. We refer to section 4 for details on this issue. Algorithm 1 is a generic algorithm for the intersection test between two faces f_1 and f_2, that are embedded on surfaces F_1 and F_2, respectively. At first, all edges of the boundary of f_1 are tested for intersection with f_2 and vice versa (lines 1 to 6). If one of these tests is positive, we have obviously found an intersection. Otherwise, we know that each loop of the intersection curve between the surfaces F_1 and F_2 lies completely inside or completely outside of f_1 and f_2, respectively. Thus, it suffices to determine one point on each loop and to test, whether it is contained in both f_1 and f_2 (lines 7 to 10). If we find one such point, we have found an intersection, otherwise we can decide that f_1 and f_2 do not intersect.

The test in lines 2 and 5 of algorithm 1 consists of two steps. At first, the curve on which the edge is embedded is intersected with the surface containing the face. Then, for each intersection point (if any) one has to check whether it is contained in both the edge and the face. Thus, the following four tasks have to be performed.

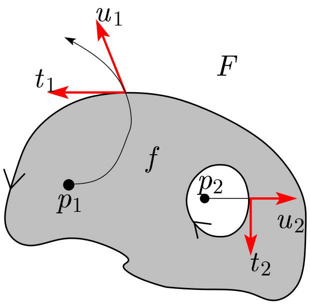
1. Test whether a point on a curve is contained in an edge embedded on that curve.
2. Test whether a point on a surface is contained in a face embedded on that surface.
3. Compute the points of intersection between a curve and a surface.
4. Compute (at least) one point on each loop of the intersection curve between two surfaces.

We will now briefly describe generically how to solve the first two of these tasks. More details as well as descriptions of the remaining two tasks will be given in the next sections, where we will have a closer look at special classes of objects.

Point-on-Edge-Test. Let the curve C be parameterized, and let the edge e on C be defined by a parameter interval $[a, b]$. Additionally we are given a point $\boldsymbol{p}$ on C. It is easy to decide whether e contains $\boldsymbol{p}$, provided that we know the parameter t of $\boldsymbol{p}$ on C. We simply have to check the condition $a \leq t \leq b$. For the case when we do not know the parameter t we refer to the sections below dealing with the special classes of objects.

Point-in-Face-Test. Let the face f be embedded on the surface F, and let $\boldsymbol{p}$ be a point on F. The boundary of f is oriented in such a way that, if one looks at f in the opposite direction of the face normal, then the interior is on the left-hand side of the oriented boundary. See figure 1 as

Figure 1 Point-In-Face-Test



$\det(\boldsymbol{u}_1, \boldsymbol{t}_1, \boldsymbol{n}_1) > 0$
$\Rightarrow \boldsymbol{p}_1$ inside

$\det(\boldsymbol{u}_2, \boldsymbol{t}_2, \boldsymbol{n}_2) < 0$
$\Rightarrow \boldsymbol{p}_2$ outside

an example. The normal in the figure is supposed to point out of the drawing plane. In order to check whether $\boldsymbol{p}$ lies inside or outside of f, we follow a continuous path within F from $\boldsymbol{p}$ to the boundary of f. We call this path a ray, because this procedure is analogous to the ray-shooting method in the case of polygons. We compute the tangent $\boldsymbol{u}$ to the ray, the tangent $\boldsymbol{t}$ to the boundary of f and the normal $\boldsymbol{n}$ of F at the point where the ray first hits a bounding edge. Because of the orientation of the boundary, the point $\boldsymbol{p}$ lies inside f iff $\det(\boldsymbol{u}, \boldsymbol{t}, \boldsymbol{n}) > 0$. Figure 1 illustrates the two cases.

We perform the point-in-face-test in 3-space rather than in the parameter space of the particular surface. The reason for this is that the curves that we allow as boundaries of the faces have a relatively simple representation in 3-space. If we change to the parameter space, then it is not guaranteed that this is still true. A conic section in 3-space for instance is in general not a conic section in the parameter space of the surface.

3. SPECIAL CASES

In this section we present three special classes of objects for which all computations in the generic algorithm can be performed efficiently and robustly, i.e., by finding the roots of polynomials of low degree. The three classes are the following:

Class 1. All faces are embedded on quadrics, and all edges are segments of conic sections. This is the natural extension of polyhedra, since the algebraic degree of the boundary elements is increased by one.

Class 2. The faces are embedded on natural quadrics and the torus, and the edges are straight line segments and circle segments. Natural quadrics are planes (as a special case), spheres, circular cones and circular cylinders. Objects of this class are the simplest ones that typically occur in CAD applications.

Class 3. The faces are embedded on quadrics, and the edges are embedded on intersection curves between quadrics or on cubic spline curves, which are often used to approximate the intersection curves. We will also briefly discuss what happens if we extend this class by the torus. Class 3 is closed under boolean operations.

When considering the torus as a surface type we always assume that the major radius is strictly larger than the minor radius in order to avoid self-intersections.

3.1 Class 1

We will show that for objects of class 1 the degrees of all polynomials occurring in algorithm 1 of section 2 are at most four. This means that the four tasks mentioned in that section can be solved by finding the roots of polynomials of degree at most four.

Point-on-Edge-Test. As stated in section 2, this task is trivial provided that the curve parameter of the query point is known. We will see that this parameter is automatically computed when we intersect the curve with a surface.

Point-in-Face-Test. Let f be a face that lies on a quadric Q, and let $\boldsymbol{p}$ be a point on Q. We must show that we can find a ray in Q from $\boldsymbol{p}$ to the boundary of f such that the intersection points between this ray and any edge of the boundary can be computed using only polynomials of sufficiently low degree.

Case 1: If Q is a ruled quadric, i.e. it has a parameterization of the form

$$\boldsymbol{x}(s,t) = \boldsymbol{a}(t) + s\boldsymbol{r}(t),$$

then we compute the parameters s_0 and t_0 such that $\boldsymbol{p} = \boldsymbol{x}(s_0, t_0)$ and define the ray starting at $\boldsymbol{p}$ by

$$\boldsymbol{l}(\lambda) = \boldsymbol{p} + \lambda \boldsymbol{r}(t_0)$$

for $\lambda \geq 0$. We compute the intersection points between this ray and the boundary of f. Since the boundary consists of conic sections, this can be done by solving only linear and quadratic equations. The intersection point with minimal positive λ-parameter is the first point where the ray hits the boundary of f. Unless this point is a tangential intersection point, we use the determinant criterion described in section 2 to decide whether $\boldsymbol{p}$ lies inside or outside of f. Otherwise, we proceed as in case 2.

Case 2: If Q is not ruled or we did not succeed in case 1, then we choose an arbitrary point $\boldsymbol{q}$ on the boundary of f. It is always possible to define a plane P through $\boldsymbol{p}$ and $\boldsymbol{q}$ such that the intersection curve between P and Q connects $\boldsymbol{p}$ and $\boldsymbol{q}$. We use that intersection curve, which is a conic section, to define the ray from $\boldsymbol{p}$ to $\boldsymbol{q}$. Let P be given by the parameterization

$$\boldsymbol{x}(s,t) = \boldsymbol{a} + s\boldsymbol{u} + t\boldsymbol{v},$$

with $|\boldsymbol{u}| = |\boldsymbol{v}| = 1$ and $\boldsymbol{u}^T\boldsymbol{v} = 0$. By inserting this into the implicit form defining Q, we obtain an implicit form for the intersection curve in the parameter space of P. By applying a principle-axis transformation on this implicit form, we can classify the conic section and parameterize it appropriately. We choose the parameterization $\boldsymbol{c}(t)$ in such a way that $\boldsymbol{c}(0) = \boldsymbol{p}$ and $\boldsymbol{c}(t) = \boldsymbol{q}$ for some $t > 0$. We intersect P with the boundary of f in order to compute the points of intersection between the ray and the boundary. We can use the parameterization of the ray to determine the t-parameters of these points. The point with minimal positive t-parameter is the first intersection point between the ray and the boundary. Now we use the determinant criterion in order to decide whether $\boldsymbol{p}$ lies inside or outside of f.

Curve-Surface-Intersection. Let C be a conic section, and let Q be a quadric. We want to compute the curve param-

ellipse	$\boldsymbol{x}(t) = [1-t^2, 2t, 1+t^2]^T$
hyperbola	$\boldsymbol{x}(t) = [1+t^2, 2t, 1-t^2]^T$
parabola	$\boldsymbol{x}(t) = [t, t^2, 1]^T$

Table 1: Polynomial parameterizations of conic sections in normal form in the projective plane.

eters of the intersection points between C and Q. We may assume that we have a polynomial parameterization $\boldsymbol{c}(t)$ of C in homogeneous coordinates. Table 1 shows that the components of $\boldsymbol{c}$ have degree at most two. Let the quadric Q be given by

$$\boldsymbol{x}^T \mathbf{A} \boldsymbol{x} = 0, \quad \text{where} \quad \mathbf{A} = \begin{bmatrix} \overline{\mathbf{A}} & \boldsymbol{a} \\ \boldsymbol{a}^T & a_0 \end{bmatrix} \tag{1}$$

for a symmetric (3×3)-matrix $\overline{\mathbf{A}}$. Inserting $\boldsymbol{c}(t)$ into (1) yields a polynomial of degree at most four in t. The roots of this polynomial are the parameters of the intersection points between C and Q.

Surface-Surface-Intersection. Now we must show that we can compute (at least) one point on each loop of the intersection curve between two quadrics, and that we can do so by finding the roots of polynomials of degree at most four. Let A and B be two quadrics defined by the (4×4)-matrices $\mathbf{A}$ and $\mathbf{B}$ as in (1). The following theorem, which is proved in [10] is the basis of our approach.

THEOREM 1. *The intersection curve between two arbitrary quadrics lies on a plane, a pair of planes, a hyperbolic or parabolic cylinder, or a hyperbolic paraboloid.*

The key observation for the proof of this theorem is that any two quadrics $Q(\lambda_1)$ and $Q(\lambda_2)$ in the pencil

$$Q(\lambda) = A - \lambda B \tag{2}$$

intersect in the same curve. The proof of theorem 1 shows that there is a root of the degree three polynomial $\det(\overline{\mathbf{A}} - \lambda\overline{\mathbf{B}})$, such that $Q(\lambda)$ is one of the quadrics mentioned in the theorem. Once this λ is found, we intersect the quadric A with $Q(\lambda)$ instead of B. The next observation is that the quadrics mentioned in theorem 1 are ruled quadrics that can be parameterized in the form

$$\boldsymbol{x}(s,t) = \boldsymbol{p}(t) + s\boldsymbol{r}(t), \tag{3}$$

where $\boldsymbol{p}$ and $\boldsymbol{r}$ are homogeneous polynomial vectors in the variable t. It can easily be shown that these parameterizations have the property

$$\deg(\boldsymbol{p}) + \deg(\boldsymbol{r}) \le 2. \tag{4}$$

Inserting this parameterization into the implicit form (1) for quadric A yields a quadratic equation for s:

$$\begin{aligned} \alpha(t)s^2 + \beta(t)s + \gamma(t) &= 0 \quad \text{with} \\ \alpha(t) &= \boldsymbol{r}(t)^T \mathbf{A} \boldsymbol{r}(t), \\ \beta(t) &= 2\boldsymbol{p}(t)^T \mathbf{A} \boldsymbol{r}(t) \quad \text{and} \\ \gamma(t) &= \boldsymbol{p}(t)^T \mathbf{A} \boldsymbol{p}(t). \end{aligned} \tag{5}$$

The discriminant of this equation as a polynomial in t is $D = \beta(t)^2 - 4\alpha(t)\gamma(t)$ and because of (4) we have

$$\deg(D) \le 2(\deg(\boldsymbol{p}) + \deg(\boldsymbol{r})) \le 4.$$

If we solve equation (5) for s we obtain

$$s(t) = \frac{-\beta(t) \pm \sqrt{D(t)}}{2\alpha(t)}. \tag{6}$$

Inserting this into the parameterization (3) gives us a parameterization of the intersection curve between the quadrics. We can find the intervals for t on which this curve is defined, by solving the equation $D(t) = 0$. In each such interval we choose a value for t and compute two points on the curve (one using the minus sign and one using the plus sign in (6)). In this way we compute at least one point on each loop of the intersection curve.

3.2 Class 2

In this section we show that also for the objects in class 2 the polynomials to be solved in algorithm 1 are of degree at most four. As before, we will have a closer look at the four tasks to be solved.

Point-on-Edge-Test. As before we will know the curve parameter of the query point whenever we must perform this test. So the test is trivial again.

Point-in-Face-Test. Let f be a face embedded on the surface F, and let $\boldsymbol{p}$ be a point on F. If F is a quadric, then we know from the previous section how to perform this test. So let F be a torus. Then all edges of f are circle segments. Let $\boldsymbol{c}$ be the center and $\boldsymbol{n}$ the normal of the main circle of the torus. We denote the minor and major radii by r and R, respectively. Moreover, let $\boldsymbol{u}$ and $\boldsymbol{v}$ be two unit vectors that are perpendicular to each other such that $\boldsymbol{n} = \boldsymbol{u} \times \boldsymbol{v}$ holds. A parameterization of the torus is then given by

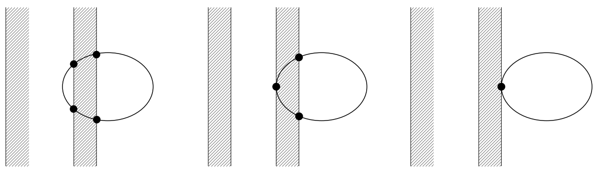
$$\boldsymbol{x} = \boldsymbol{c} + (R + r\cos\psi)(\cos\varphi\boldsymbol{u} + \sin\varphi\boldsymbol{v}) + r\sin\psi\boldsymbol{n}. \tag{7}$$

For any point $\boldsymbol{a}$ on the torus we denote the parameters of $\boldsymbol{a}$ by $\varphi_{\boldsymbol{a}}$ and $\psi_{\boldsymbol{a}}$. These parameters can easily be computed using inverse trigonometric functions. Let $\boldsymbol{q}$ be an arbitrarily chosen point on the boundary of f. Now we must send a ray on the torus from $\boldsymbol{p}$ to $\boldsymbol{q}$. This ray will follow the cross sectional circle through $\boldsymbol{p}$ until it reaches the profile circle through $\boldsymbol{q}$. Then it will follow this circle until it reaches $\boldsymbol{q}$. We first compute $(\varphi_{\boldsymbol{p}}, \psi_{\boldsymbol{p}})$ and $(\varphi_{\boldsymbol{q}}, \psi_{\boldsymbol{q}})$ in the range of $[-\pi, \pi)$. If $\varphi_{\boldsymbol{q}} < \varphi_{\boldsymbol{p}}$ we replace $\varphi_{\boldsymbol{q}}$ by $\varphi_{\boldsymbol{q}} + 2\pi$, and similarly for $\psi_{\boldsymbol{q}}$. In order to compute the intersection points between the ray and the boundary of f, we consider the plane defined by

$$\boldsymbol{x}(\lambda, \mu) = \boldsymbol{p} + \lambda\boldsymbol{r} + \mu\boldsymbol{n},$$

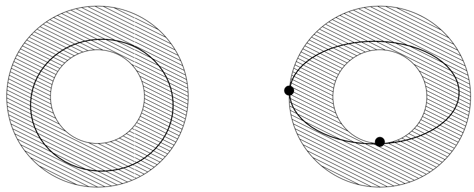
with $\boldsymbol{r} = \cos\varphi_{\boldsymbol{p}}\boldsymbol{u} + \sin\varphi_{\boldsymbol{p}}\boldsymbol{v}$. This plane contains the first part of the ray. To compute the points of intersection between the plane and the boundary of f, we need to solve a quadratic equation. For each such point $\boldsymbol{a}$ we compute $\psi_{\boldsymbol{a}}$, and if this value is less than $\psi_{\boldsymbol{p}}$ we add 2π to it. If the smallest of these values is less than or equal to $\psi_{\boldsymbol{q}}$, then $\boldsymbol{a}$ lies on the first part of the ray and we decide whether $\boldsymbol{p}$ lies inside or outside of f using the determinant criterion. Otherwise, we proceed similarly for the second part of the ray.

Figure 2 Intersection of the main circle of a torus and the C-space obstacle of a cylinder. The intersection is not the entire main circle.



(a) (b) (c)

Figure 3 Intersection of the main circle of a torus and the C-space obstacle of a cylinder. The intersection is the entire main circle.



(a) (b)

Curve-Surface-Intersection. We know from the previous section that we need to solve only polynomials of degree at most four to compute the intersection between conic sections and quadric surfaces, so we can concentrate on the case that F is a torus. Let C be a straight line or a circle. The implicit representation of the torus is

$$\left((\boldsymbol{x}-\boldsymbol{c})^2+R^2-r^2\right)^2-4R^2\left((\boldsymbol{x}-\boldsymbol{c})\times\boldsymbol{n}\right)^2=0. \qquad (8)$$

If C is a straight line with the parameterization $\boldsymbol{x}(t) = \boldsymbol{a}+t\boldsymbol{r}$, then inserting $\boldsymbol{x}(t)$ into (8) leads to a polynomial of degree four in t. The roots of this polynomial are the parameters of the intersection points on C. For the case of C being a circle, we refer the reader to [7], where it is shown that computing the points of intersection between a circle and a torus can also be reduced to finding the roots of a polynomial of degree four.

Surface-Surface-Intersection. If the surfaces are both quadrics, then we can apply the results of section 3.1. Now let one of the surfaces be a torus. We briefly describe the so-called configuration space approach used in [7] to determine points on the intersection curve between a natural quadric and a torus, as well as between two tori. The idea is the following. Given two surfaces, say a torus T and a cylinder C, one of them is considered as the envelope surface of a moving ball and the other one as an obstacle. Let r be the minor and R the major radius of the torus. The cylinder is given by its axis through the point $\boldsymbol{p}$ with direction $\boldsymbol{u}$ and its radius ρ. Let us assume that $r < \rho$. In this case we consider the torus as the envelope surface of a ball with radius r moving along a circle with radius R in the (x, y)-plane centered at the origin. Now we replace this moving ball by its center, and every point of the cylinder by a ball of radius r. This means that the torus is shrunk to a circle, whereas the cylinder surface is blown up to its so-called C-space obstacle which is bounded by the r-offset-surface of the original cylinder. This offset-surface consists of two co-axial cylinders with radii $\rho-r$ and $\rho+r$. The black dots in figures 2 and 3 show points of intersection between the circle and the two co-axial cylinders. They result from positions where the moving ball touches the original cylinder surface tangentially. The points on the circle that lie in the interior of the C-space obstacle result from positions where the ball properly intersects the cylinder surface. Let S be a segment of the intersection between the main circle of T and the C-space obstacle. Then the following can be shown.

- If S is not the entire main circle and not a single point, then S results from a closed loop of the intersection curve. This is the case for the two segments sketched in figure 2(a). If S moreover touches the boundary of the C-space obstacle tangentially in k points, then the loop has k singular points. This is sketched in figure 2(b).
- If S is a single point, then it results from an isolated point of the intersection curve. This is sketched in figure 2(c).
- If S is the entire main circle and does not touch the boundary of the C-space obstacle, then the intersection curve consists of two closed loops (figure 3(a)).
- If S is the entire main circle and touches the boundary of the C-space obstacle in $k > 0$ points, then the intersection curve consists of one closed loop with k singular points (figure 3(b)).

Computing the intersection of the main circle of T with the C-space obstacle of C means computing the points of intersection between a circle and two cylinders. We know from the previous sections that this can be reduced to finding the roots of polynomials of degree at most four. Once we have computed this intersection, we must construct one point on each loop of the intersection curve.

- In the cases shown in figure 2 and figure 3(b), we first choose one of the intersection points for each segment. If for any segment there is a tangential intersection point, then we choose this point. Let $\boldsymbol{q}$ be such a point. We know that $\boldsymbol{q}$ results from a situation where the moving ball touches the cylinder surface. We compute this contact point $\boldsymbol{q}'$. Then we determine the intersection points between the line through $\boldsymbol{q}'$ with direction $\boldsymbol{u}$ and the torus. As a point on the loop, we can choose that intersection point that is closest to $\boldsymbol{q}'$.
- In the case shown in figure 3(a), we may choose any point on the main circle. Let $\boldsymbol{q}$ be such a point. We compute the two intersection points between the cross-sectional circle centered at $\boldsymbol{q}$ and the cylinder surface. These points lie on the two loops of the intersection curve.

If the radius ρ of C is equal to r, then the inner cylinder of the C-space obstacle degenerates to a straight line. In this case one has to be more careful, but the general idea still works. If $r > \rho$, then we consider the torus as the obstacle and the cylinder as the envelope surface of a moving ball. We shrink the cylinder to a straight line and replace the

plane	$\hat{\boldsymbol{x}}(s,t) = [t, s, 0]^T$
hyperbolic cylinder	$\hat{\boldsymbol{x}}(s,t) = \left[t, \frac{1}{t}, s\right]^T$
parabolic cylinder	$\hat{\boldsymbol{x}}(s,t) = [t, s, -t^2]^T$
hyperbolic paraboloid	$\hat{\boldsymbol{x}}(s,t) = [t, s, st]^T$

Table 2: Parametric forms of the quadrics mentioned in theorem 1.

torus by its C-space obstacle, which is bounded by the $\pm\rho$-offset-surfaces of the torus. Then we proceed similar to the case when $r < \rho$.

The configuration space approach described above can be applied to all intersection tests between natural quadrics and tori, as well as between two tori. In [7] it is described how this can be done and that all polynomial equations that have to be solved have degree four at most.

3.3 Class 3

In this section we show that all computations for the collision test between two objects of class 3 can be performed by solving polynomial equations of degree eight at most. At the end of the section we will briefly discuss what happens if we extend this class by torus surfaces.

Point-on-Edge-Test. Let $\boldsymbol{q}$ be a point, and let the edge be embedded on the curve C. If C is a spline curve, then we know the curve parameter of $\boldsymbol{q}$ from the curve-surface intersection-test. So it is trivial to decide whether the edge contains $\boldsymbol{q}$. If C is a quadric intersection curve, then the method that we use for the curve-surface intersection-test does not produce the curve parameter of the intersection points. So we must compute the parameter of $\boldsymbol{q}$ in that case. We assume that the parameterization of C is given by inserting equation (6) of section 3.1 into the parametric representation of a quadric mentioned in theorem 1 of that section. We also assume that that quadric is given by $\boldsymbol{x}(s,t) = \mathbf{S}\hat{\boldsymbol{x}}(s,t)+\boldsymbol{c}$, for a transformation matrix $\mathbf{S}$, a vector $\boldsymbol{c}$, and $\hat{\boldsymbol{x}}(s,t)$ being one of the parameterizations in table 2. The parameterization of C is $\boldsymbol{y}(t) = \boldsymbol{x}(s(t),t)$. Since we know that $\boldsymbol{q}$ lies on C, we can write $\hat{\boldsymbol{x}}(s(t),t) = \mathbf{S}^{-1}(\boldsymbol{q}-\boldsymbol{c})$. Thus, the curve parameter of $\boldsymbol{q}$ is the first component of $\boldsymbol{S}^{-1}(\boldsymbol{q}-\boldsymbol{c})$ (cf. table 2).

Point-in-Face-Test. Let $\boldsymbol{p}$ be the point to be tested for containment in the face f that is embedded on the quadric Q, and let $\boldsymbol{q} \neq \boldsymbol{p}$ be a point on the boundary of f. We saw in section 3.1 that we can always construct a ray from $\boldsymbol{p}$ to $\boldsymbol{q}$ that lies in Q by intersecting Q with a suitable plane. Intersecting the ray with the boundary of f then means computing the intersections between the plane containing the ray and the boundary of f. Let C be a curve containing an edge of the boundary of f. We must show that the intersection between C and a plane can be easily computed. We make a case distinction with respect to the type of C.

Case 1: C is a quadric intersection curve. Let the two quadrics defining C be given by $f(\boldsymbol{x}) = 0$ and $g(\boldsymbol{x}) = 0$, respectively, where f and g are quadratic polynomials in the components of $\boldsymbol{x}$. Let the plane be defined parametrically by

$$\boldsymbol{x}(s,t) = \boldsymbol{a} + s\boldsymbol{u} + t\boldsymbol{v}.$$

We insert this into f and g and obtain a system of two quadratic polynomial equations in the variables s and t. Eliminating s by means of resultants leads to a degree four polynomial in t (see, for example, [2] for details on resultants). We compute the roots of these polynomials and insert them into $f(s,t) = 0$ in order to determine a quadratic equation for the corresponding value of s.

Case 2: C is a cubic spline curve. Then C can be written as a piecewise polynomial function

$$\boldsymbol{x}(t) = \begin{cases} \boldsymbol{c}_1(t) & \text{if } 0 \le t < u_1, \\ \vdots & \\ \boldsymbol{c}_k(t) & \text{if } u_{k-1} \le t < 1, \end{cases} \tag{9}$$

where the components of the $\boldsymbol{c}_i$ are polynomials of degree three. Let the plane be given by $\boldsymbol{n}^T\boldsymbol{x} - n_0 = 0$. We insert each of the $\boldsymbol{c}_i$ into that equation and obtain a polynomial of degree three for each piece of C. For the ith piece we compute the roots of the corresponding polynomial in the interval $[t_{i-1}, t_i)$. In this way, we find all points of intersection between the plane and C.

Curve-Surface-Intersection. Let Q be a quadric surface defined by the quadratic form $f(\boldsymbol{x}) = 0$. As in the previous paragraph, we make a case distinction according to the curve type.

Case 1: C is an intersection curve between two quadrics that are defined by the quadratic forms $g(\boldsymbol{x}) = 0$ and $h(\boldsymbol{x}) = 0$. Finding the points of intersection between C and Q is the same as finding the intersection points between the three quadrics. This means finding the roots of the system of polynomial equations

$$f(\boldsymbol{x}) = g(\boldsymbol{x}) = h(\boldsymbol{x}) = 0. \tag{10}$$

We write $\boldsymbol{x} = [x, y, z]^T$. In [1], it is shown that this system can be reduced to a univariate polynomial of degree at most eight using multivariate resultants. The multivariate resultant is a polynomial in the coefficients of a system of k homogeneous polynomial equations in k variables. This polynomial vanishes if and only if the system of homogeneous equations has a non-trivial solution (in the complex numbers). The equations in (10) are not homogeneous. So one regards one of the variables, say x, as an intermediate constant and introduces a new variable w to homogenize the polynomials (now considered as polynomials in w, y and z). Now the multivariate resultant is a polynomial in the coefficients of these homogeneous equations and thus it is also a polynomial in x. [1] proves that the degree of this resultant as a polynomial in x is at most eight. Now the roots of this polynomial are computed using numerical methods. These roots are inserted into (10) in order to obtain three quadratic equations in two variables for each root. These can be reduced to univariate polynomials of degree four by means of 'standard' resultants.

Case 2: C is a cubic spline curve. We write C as a piecewise polynomial function as in (9). We insert each piece $\boldsymbol{c}_i(t)$ into the quadratic form f defining the quadric Q and obtain polynomial equations of degree six. We compute the roots in the intervals $[t_{i-1}, t_i)$ and in this way we find all points of intersection between C and Q.

Surface-Surface-Intersection. This test is exactly the same as the one for objects of class 1 described in section 3.1.

4. BOUNDING VOLUMES

In this section we describe how to compute bounding volumes for a given set of faces. We consider spheres and arbitrarily oriented boxes as bounding volumes, both of which can be very efficiently tested for intersection. Another important property of these types of bounding volumes is that they do not have to be recomputed as the objects rotate, as it is the case for axis aligned boxes or fixed direction hulls.

We discuss the problem for objects that are bounded by quadric surface patches, torus patches and conic arcs.

The general idea behind our approach is to use local optimization. We minimize the volume of the enclosing sphere or box over all possible centers or orientations, respectively.

Let $\mathcal{F}$ be the set of faces for which the bounding volume is to be computed. Further, let $\mathcal{E}$ denote the set of edges of all faces of $\mathcal{F}$ and $\mathcal{V}$ the set of vertices.

4.1 Enclosing sphere

The maximal distance between a point $\boldsymbol{p}$ and any point of the faces, edges and vertices is given by

$$\begin{aligned} d(\boldsymbol{p}) = \max\{&\max_{f\in\mathcal{F}}\max_{x\in f}|\boldsymbol{p}-\boldsymbol{x}|, \\ &\max_{e\in\mathcal{E}}\max_{x\in e}|\boldsymbol{p}-\boldsymbol{x}|,\ \max_{x\in\mathcal{V}}|\boldsymbol{p}-\boldsymbol{x}|\}. \end{aligned} \tag{11}$$

The radius $r(\boldsymbol{c})$ of the bounding sphere with center $\boldsymbol{c}$ is given by $d(\boldsymbol{c})$. Since the volume of a sphere is a strictly monotone function of the radius, it is sufficient to minimize the radius in order to minimize the volume.

In the next two subsections we show how to compute the expressions $\max_{x\in e}|\boldsymbol{p}-\boldsymbol{x}|$ and $\max_{x\in f}|\boldsymbol{p}-\boldsymbol{x}|$ for an edge $e\in\mathcal{E}$ and a face $f\in\mathcal{F}$, respectively.

Maximal distance point-edge. Let $e\in\mathcal{E}$ be an edge embedded on the curve C. We compute the local maxima of the function

$$g_{\boldsymbol{p}}(\boldsymbol{x}) = |\boldsymbol{p}-\boldsymbol{x}|, \qquad \boldsymbol{x}\in C. \tag{12}$$

For each local maximum we check whether the corresponding point is part of the edge e or not (cf. section 3).

Plugging the rational parameterization of a conic C into equation (12) results in a quartic polynomial.

Maximal distance point-face. Let $f\in\mathcal{F}$ be a face embedded on the surface F. We compute the local maxima of the function

$$g_{\boldsymbol{p}}(\boldsymbol{x}) = |\boldsymbol{p}-\boldsymbol{x}|, \qquad x\in F. \tag{13}$$

For each local maximum we check whether the corresponding point is contained in face f or not (cf. section 3).

Function (13) has no local maxima if F is an unbounded surface, e.g. a plane, a cone or a cylinder. If F is a sphere given by $|\boldsymbol{x}-\boldsymbol{c}| = r$ with $\boldsymbol{c}\neq\boldsymbol{p}$, there is exactly one local maximum $\boldsymbol{x}_{max} = \boldsymbol{c} + r\frac{\boldsymbol{c}-\boldsymbol{p}}{|\boldsymbol{c}-\boldsymbol{p}|}$. If $\boldsymbol{c}=\boldsymbol{p}$, all points on the sphere have distance r from the point $\boldsymbol{p}$. If F is an ellipsoid, the maximal distance between $\boldsymbol{p}$ and F can be described by the roots of a polynomial of degree six; see [4] for details. If F is a torus with center $\boldsymbol{c}$, normal $\boldsymbol{n}$ with $|\boldsymbol{n}|=1$ and radii R and r with $R>r$, the local maximum of function (13) can be computed as follows:

$$\begin{aligned} \boldsymbol{x}_{max} &= \tilde{\boldsymbol{x}}_{max} - r\frac{\boldsymbol{p}-\tilde{\boldsymbol{x}}_{max}}{|\boldsymbol{p}-\tilde{\boldsymbol{x}}_{max}|}, \\ \tilde{\boldsymbol{x}}_{max} &= \boldsymbol{c} - R\frac{\boldsymbol{p}-\boldsymbol{c}-\boldsymbol{n}^T(\boldsymbol{p}-\boldsymbol{c})\boldsymbol{n}}{|\boldsymbol{p}-\boldsymbol{c}-\boldsymbol{n}^T(\boldsymbol{p}-\boldsymbol{c})\boldsymbol{n}|}. \end{aligned}$$

Smallest enclosing sphere. Our aim is to compute the optimal sphere center $\boldsymbol{c}^*$, such that its radius $r(\boldsymbol{c}^*)$ is the minimum of $d(\boldsymbol{p})$.

LEMMA 1. *The maximal distance $d(\boldsymbol{p})$ is a convex function of $\boldsymbol{p}$.*

Since the function $d(\boldsymbol{p})$ is convex, there is exactly one local minimum. Thus it is sufficient to compute the local minimum of $d(\boldsymbol{p})$ in order to compute the optimal bounding sphere.

We use two local optimization methods, both presented in [12], the *downhill simplex method* due to Nelder and Mead and *Powell's method.*

4.2 Enclosing boxes

In order to specify the orientation of an enclosing box we use two different representations. A set of three orthonormal vectors $(\boldsymbol{d_1},\boldsymbol{d_2},\boldsymbol{d_3})$ defines the axes of the box. The second representation uses three angles $(\alpha,\beta,\gamma)\in[-\pi,\pi]\times[-\frac{\pi}{2},\frac{\pi}{2}]\times[-\pi,\pi]$. Those angles are interpreted as Z-Y-X Euler angles. The columns of the corresponding rotation matrix $R_{ZYX}(\alpha,\beta,\gamma)$ are the vectors $\boldsymbol{d_1},\boldsymbol{d_2}$ and $\boldsymbol{d_3}$.

Let $\boldsymbol{d}$ denote a direction, $|\boldsymbol{d}|=1$. The extent in direction $\boldsymbol{d}$ is given by

$$\begin{aligned} e(\boldsymbol{d}) &= e_{max}(\boldsymbol{d}) - e_{min}(\boldsymbol{d}) \\ e_{max}(\boldsymbol{d}) &= \max\{\max_{f\in\mathcal{F}}\max_{x\in f}\boldsymbol{d}^T\boldsymbol{x}, \max_{e\in\mathcal{E}}\max_{x\in e}\boldsymbol{d}^T\boldsymbol{x}, \max_{x\in\mathcal{V}}\boldsymbol{d}^T\boldsymbol{x}\} \\ e_{min}(\boldsymbol{d}) &= \min\{\min_{f\in\mathcal{F}}\min_{x\in f}\boldsymbol{d}^T\boldsymbol{x}, \min_{e\in\mathcal{E}}\min_{x\in e}\boldsymbol{d}^T\boldsymbol{x}, \min_{x\in\mathcal{V}}\boldsymbol{d}^T\boldsymbol{x}\}. \end{aligned}$$

A corner point of the enclosing box is given by the solution of $\boldsymbol{d}_i^T\boldsymbol{x} = e_{min}(\boldsymbol{d}_i)$ for $i=1,2,3$. The edges $\boldsymbol{a}_1$, $\boldsymbol{a}_2$ and $\boldsymbol{a}_3$ of this box are given by $\boldsymbol{a}_i = e(\boldsymbol{d}_i)\,\boldsymbol{d}_i$ for $i=1,2,3$, and the volume can be computed as $V(\alpha,\beta,\gamma)=\prod_{i=1}^3 e(\boldsymbol{d}_i)$.

In the next two subsections we show, how to compute the expressions $\max_{x\in e}\boldsymbol{d}^T\boldsymbol{x}$, $\max_{x\in f}\boldsymbol{d}^T\boldsymbol{x}$, $\min_{x\in e}\boldsymbol{d}^T\boldsymbol{x}$ and $\min_{x\in f}\boldsymbol{d}^T\boldsymbol{x}$ for an edge $e\in\mathcal{E}$ and a face $f\in\mathcal{F}$.

Extent of an edge. Let $e\in\mathcal{E}$ be an edge embedded on the curve C. We compute the local extrema of the function

$$g_{\boldsymbol{d}}(\boldsymbol{x}) = \boldsymbol{d}^T\boldsymbol{x}, \qquad x\in C. \tag{14}$$

For each local extremum we check whether the corresponding point is part of the edge e or not (cf. section 3).

We do not need to consider straight lines since the endpoints generate the correct values for $\boldsymbol{d}^T\boldsymbol{x}$. It is easy to see that if C is a conic, then computing these extrema can be done by solving quadratic equations.

Extent of a face. Let $f\in\mathcal{F}$ be a face embedded on the surface F. We compute the local extrema of the function

$$g_{\boldsymbol{d}}(\boldsymbol{x}) = \boldsymbol{d}^T\boldsymbol{x}, \qquad x\in F. \tag{15}$$

For each local extremum we check whether the corresponding point is contained in the face f or not (cf. section 3).

We do not need to consider planes, cones or cylinders, since there are always edges or points that contribute the same value for $\boldsymbol{d}^T\boldsymbol{x}$. If F is a sphere given by $|\boldsymbol{x}-\boldsymbol{c}| = r$, the local extrema of (15) are $\boldsymbol{c} \pm r\boldsymbol{d}$. For all other quadric surfaces we use the following general approach. If F is a quadric given in the form (1), then the direction of the face normal $\boldsymbol{n}(\boldsymbol{x})$ at point $\boldsymbol{x}$ can be computed as $\boldsymbol{n}(\boldsymbol{x}) = \overline{\mathbf{A}}\boldsymbol{x}+\boldsymbol{a}$. For a point $\boldsymbol{x}$ with extremal extent, it holds that $\boldsymbol{n}(\boldsymbol{x}) = \lambda\boldsymbol{d}$. Solving this equation for $\boldsymbol{x}$ and substituting into (1) yields a quadratic equation in λ.

Let F be a torus given by equation (7). Then the local extrema of (15) can be computed by first determining the extremal points on the main circle of F, which leads to a quadratic equation, and then adding the vector $\pm r\boldsymbol{d}$ to each of these points. This leads to the extrema

$$\boldsymbol{x}_{\substack{max\\min}} = \boldsymbol{c} \pm R\frac{\boldsymbol{d}-(\boldsymbol{d}^T\boldsymbol{n})\boldsymbol{n}}{\sqrt{1-(\boldsymbol{d}^T\boldsymbol{n})^2}} \pm r\boldsymbol{d}.$$

Smallest enclosing box. Now we focus on minimizing the volume of the enclosing box. Our aim is to compute an orientation $(\alpha^*, \beta^*, \gamma^*)$, such that the volume of the corresponding enclosing box is locally minimal. As in the case of the smallest enclosing sphere, we use local optimization methods to minimize the volume $V(\alpha, \beta, \gamma)$. But in contrast to the smallest enclosing sphere, the objective function is no longer convex. Thus we only get boxes with locally minimal volume.

The quality of the computed local minimum can be improved by repeating the optimization process with different starting points. Thus it is likely that the optimization method computes different local minima and one chooses the minimum with the smallest function value.

5. CONCLUSIONS

In the previous sections we described a generic algorithm for the static intersection test between two solids with curved surfaces. We showed that all computational tasks can be reduced to polynomial equations of low degree by restricting oneself to certain classes of objects. Since there are closed-form solutions for polynomial equations of degree at most four, these equations can be solved robustly and efficiently. Our experiments showed that using the explicit formulae for the roots of these polynomials is sufficiently accurate and much faster than numerical methods. For polynomials of higher degree we use the eigenvalue method (see [5]). This method is also very efficient and numerically robust since the degrees of the arising polynomials are at most eight.

We also presented a method for computing optimal bounding volumes for a given set of surfaces. Since the computation of the bounding volumes is done in a preprocessing step, the time consumption of the methods used there is not a problem. With a heuristic similar to the one used in [6] to divide the set of surfaces into two proper subsets we are able to build up a hierarchy of bounding volumes. This structure can be used to speed up the collision detection by quickly excluding large parts of the object from the computation.

6. ADDITIONAL AUTHORS

Additional author: Christian Lennerz (Max-Planck-Institut für Informatik, Saarbrücken, Germany, email: lennerz@mpi-sb.mpg.de)

7. REFERENCES

[1] E.-W. Chionh, R.N. Goldman, and J. R. Miller. Using multivariate resultants to find the intersection of three quadric surfaces. *ACM Transactions on Graphics*, 10(4):378–400, 1991.

[2] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms.* Springer-Verlag New York, Inc., 2nd edition, 1997.

[3] T. Dokken. *Aspects of Intersection Algorithms and Approximation.* Ph.D. thesis, University of Oslo, July 1997.

[4] D. H. Eberly. *3D game engine design : a practical approach to real-time computer graphics.* Morgan Kaufmann, 2001.

[5] S. Fortune. An iterated eigenvalue algorithm for approximating the roots of a univariate polynomial. In *Internat. Symposium on Symbolic and Algebraic Computation, ISSAC 2001*, pages 121–128, 2001.

[6] S. Gottschalk, M. C. Lin, and D. Manocha. OBB-tree: A hierarchical structure for rapid interference detection. *Computer Graphics*, pages 171–180, August 1996. Proc. SIGGRAPH'96.

[7] Ku-Jin Kim. *Torus and Simple Surface Intersection Based on a Configuration Space Approach.* Ph.D. thesis, Department of Computer Science and Engineering, POSTECH, February 1998.

[8] S. Krishnan, M. Gopi, M. Lin, D. Manocha, and A. Pattekar. Rapid and accurate contact determination between spline models using ShellTrees. *Computer Graphics Forum*, 17(3):315–326, 1998.

[9] S. Krishnan and D. Manocha. Algebraic loop detection and evaluation algorithms for curve and surface interrogations. In *Proceedings of Graphics Interface*, pages 87–94, May 1996.

[10] J. Levin. A parametric algorithm for drawing pictures of solid objects composed of quadric surfaces. *Commun. ACM*, 19(10):555–563, 1976.

[11] M. Lin and S. Gottschalk. Collision detection between geometric models: a survey. In *Proc. of IMA Conference on Mathematics of Surfaces*, 1998.

[12] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes in C : the art of scientific computing.* Cambridge Univ. Press, 2nd edition, 1994.

8. REFERENCES

[1] E.-W. Chionh, R.N. Goldman, and J. R. Miller. Using multivariate resultants to find the intersection of three quadric surfaces. *ACM Transactions on Graphics*, 10(4):378–400, 1991.

[2] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms.* Springer-Verlag New York, Inc., 2nd edition, 1997.

[3] T. Dokken. *Aspects of Intersection Algorithms and Approximation.* Ph.D. thesis, University of Oslo, July 1997.

[4] D. H. Eberly. *3D game engine design : a practical approach to real-time computer graphics.* Morgan Kaufmann, 2001.

[5] S. Fortune. An iterated eigenvalue algorithm for approximating the roots of a univariate polynomial. In *Internat. Symposium on Symbolic and Algebraic Computation, ISSAC 2001*, pages 121–128, 2001.

[6] S. Gottschalk, M. C. Lin, and D. Manocha. OBB-tree: A hierarchical structure for rapid interference detection. *Computer Graphics*, pages 171–180, August 1996. Proc. SIGGRAPH'96.

[7] Ku-Jin Kim. *Torus and Simple Surface Intersection Based on a Configuration Space Approach.* Ph.D. thesis, Department of Computer Science and Engineering, POSTECH, February 1998.

[8] S. Krishnan, M. Gopi, M. Lin, D. Manocha, and A. Pattekar. Rapid and accurate contact determination between spline models using ShellTrees. *Computer Graphics Forum*, 17(3):315–326, 1998.

[9] S. Krishnan and D. Manocha. Algebraic loop detection and evaluation algorithms for curve and surface interrogations. In *Proceedings of Graphics Interface*, pages 87–94, May 1996.

[10] J. Levin. A parametric algorithm for drawing pictures of solid objects composed of quadric surfaces. *Commun. ACM*, 19(10):555–563, 1976.

[11] M. Lin and S. Gottschalk. Collision detection between geometric models: a survey. In *Proc. of IMA Conference on Mathematics of Surfaces*, 1998.

[12] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes in C : the art of scientific computing.* Cambridge Univ. Press, 2nd edition, 1994.