

Computing Planar Sections of Surfaces of Revolution with Revolute Quadric Decomposition

Jinyuan Jia^{*†} Kai Tang^{*} Ajay Joneja^{*}

The Hong Kong University of Science and Technology^{*}
Clear Water Bay, Kowloon, Hong Kong

Liaoning University of Petroleum & Chemical Technology[†]
Fushun, Liaoning Province

{csjyjia@ust.hk, mektang@ust.hk, joneja}@ust.hk}

Ki-Wan Kwok

Department of Computing

The Hong Kong Polytechnic University

Hung Hom, Kowloon, Hong Kong

cskwkwok@comp.polyu.edu.hk

Abstract

Computing the planar sections of objects is a fundamental operation in solid modeling. Subdivision method is commonly used for solving such intersection problems. In this paper, a novel revolute quadric decomposition is proposed for surfaces of revolution, which are subdivided into a set of coaxial revolute quadrics along the generatrix. This reduces the intersection problem of a plane and a surface of revolution to the intersection problem of a plane and a revolute quadric, which has robust, accurate and efficient geometric solution. Further, the intersection curves can be represented with a group of G^1 conic arcs. A new concept, valid intersection interval (VII), is introduced and a new technique, cylindrical bounding shell clipping, is proposed for efficient intersection detection for a plane and a surface of revolution. Finally, a tracing algorithm is presented for recognizing singular points and closed loops of intersection curves. Implemented examples show the robustness and effectiveness of the proposed algorithm.

1. Introduction

The plane/surface intersection problem is essential in geometric modeling and can find wide applications in CAD, CAM, VR and computer graphics, e.g. numerical control machining (milling) involves intersections of offset surfaces with a series of parallel planes to create machining paths for ball cutters. Alternatively, in Constructive Solid Geometry (CSG) modeling applications, planes are usually used to construct new objects with surfaces altogether that involve the intersection curves of planes and surfaces for representing their boundaries. Computing plane sections of surfaces is required for rendering the contours of surfaces, which is very useful in descriptive geometry, drafting and manufacturing. Developing *robust, accurate and efficient* algorithms for surface intersections remains a challenging task for many years. Surfaces of revolution are an important member of CSG library and is frequently used in various applications of CAD/CAM, VR and CG. It is necessary

to develop a robust, accurate and efficient algorithm for computing plane sections of surfaces of revolution.

In general, there are four approaches to solving surface intersection problems: *subdivision method, algebraic method, lattice method* and *tracing method*, a good summary can be found in [17]. Subdivision method is the most robust among them. Basically, subdivision method subdivides surfaces into quadrilateral/triangular meshes and reduces the surface/surface intersection problem to the quad/quad or triangle/triangle intersection problem. However, it is quite difficult to balance between the subdivision density and precision. If the triangular mesh is subdivided densely for achieving good approximation quality, it will lead to data proliferation and slow computation; if meshed sparsely, the resulting intersection curves may miss the tiny loops and generate incorrect results topologically.

As a possible remedy to the problems, quadric decomposition is proposed in [5] for rendering purpose; instead of quads or triangles, quadric decomposition subdivides the surface into piecewise quadric patches. There has been some existing work [2, 4, 5, 9, 10, 18] on how to construct a G^1 triangular quadric net. However, all of them are for general surfaces, which might not be efficient when applied to special surfaces, e.g., surfaces of revolution. There should be more efficient quadric decomposition for these special surfaces. In this paper, we present a specific *revolute quadric decomposition* for surfaces of revolution. Based on the proposed revolute quadric decomposition, a new algorithm for computing plane sections of surfaces of revolution is devised especially.

2. Related work

There are a few previous reports involving computing plane sections of surfaces of revolution. Firstly, Kim et al [13] used algebraic method to give an elegant closed form representation for the intersection curves of a plane and a surface of revolution; however, it contains square roots and have to be further approximated by a group of lower-degree rational curve segments piecewisely, since most current CAD systems can only deal with rational curves

and surfaces. Moreover, from the given representation, we cannot directly detect closed loops and singular points of the intersection curves, recognizing them still requires numerical solutions of high-degree algebraic equations.

Considering the fact that approximations cannot be avoided during intersection processing, Baciu *et al* [1] proposed a subdivision method for solving this problem, which subdivides a surface of revolution into a sequence of coaxial truncated cones along its generatrix and reduces the intersection problem of plane/surface of revolution to the intersection problem of plane/truncated cones; but it usually requires too many truncated cones to approximate surfaces of revolution for good approximation quality and produces densely G^0 piecewise conic arcs as the final plane section curves of surfaces of revolution, since truncated cone decomposition is a low-efficient quadric decomposition for surfaces of revolution.

In the realm of intersection, there are in general three existing decomposition schemes for solving intersection problems involving surfaces of revolution, namely quad/triangle decomposition [8], circle decomposition [14], and truncated cone decomposition [1] (see Fig. 1). In this paper, we propose a new quadric decomposition, the *revolute quadric* decomposition, which subdivides a surface of revolution into a sequence of G^1 coaxial revolute quadrics along its revolute axis and reduces the intersection problem of plane and surface of revolution to the intersection problem of a plane and revolute quadrics. It produces much fewer but more smoothly basic fitting elements than other three existing schemes, and the resulting intersection curves can be represented by a group of fewer G^1 piecewise conic arcs.

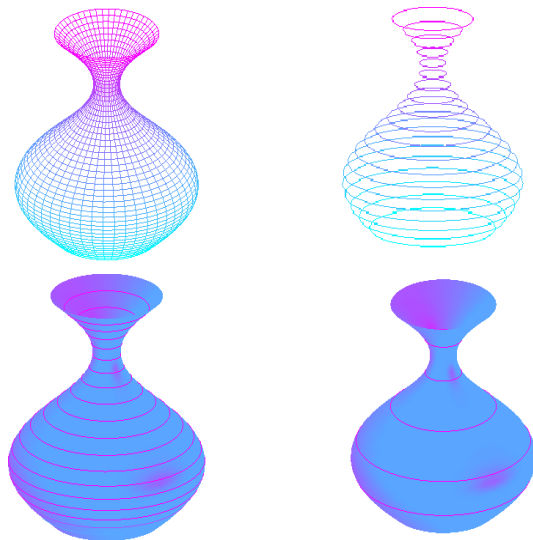


Figure 1. Four decompositions of surfaces of revolution:
 (a) quadrilateral decomposition, (b) circle decomposition, (c) truncated cone decomposition, (d) revolute quadric decomposition

Comparing with the two previous methods, our new method is based on *revolute quadric decomposition*, it possesses the following advantages:

- Fundamentally, there are known geometric methods to compute the plane sections of a revolute quadric robustly, accurately and efficiently.
- The final intersection curves can be represented by G^1 piecewise conic arcs; conic arc is a good form for contemporary CAD/CAM systems.
- Detection of the closed loops and singular points of the intersection curve becomes very easy by tracing piecewise conic arcs sequentially along the revolute axis.

3. Revolute quadric decomposition

A surface of revolution is an envelope of the generatrix about its axis. So subdividing it into a collection of revolute quadrics in 3D space is equivalent to subdividing its generatrix into a collection of coaxial conic arcs along the axis in 2D space. In this section, we propose two kinds of coaxial conic interpolators, namely the G^0 single least-squares (LS) coaxial conic fitting and the G^1 coaxial bi-conic arc spline, to approximate the generatrix for the revolute quadric decomposition of surfaces of revolution.

A conic section in the xy plane is represented as $Ax^2 + By^2 + Cxy + Dx + Ey + F = 0$ in general. It can be reduced to $Ax^2 + By^2 + Dx + F = 0$ by assuming that the revolute axis is the x -axis, and further to $Ax^2 + y^2 + Dx + F = 0$ by isolating the degenerated case of vertical line $B = 0$. There are only three freedoms A , D and F ; geometrically, they correspond to two axial radii r_a , r_b , and the x -coordinate C_x of the center of the conic section. If we assume that the axis of the surface of revolution is the x -axis, the problem becomes how to approximate the generatrix optimally with a group of coaxial conic arcs $C_i: Ax^2 + y^2 + Dx + F_i = 0$.

Many *general* conic fitting algorithms were suggested in the literature, which involve how to approximate a curve into a sequence of *general* conic arcs, but not *coaxial* conic arcs. So, the restriction ‘*coaxial*’ is too strong for any general conic arc fitting algorithms to be employable in this special problem. Therefore, it is necessary to develop a specific coaxial conic arc fitting method for subdividing the generatrix effectively.

Locally, we construct two kinds of coaxial conic arc interpolators, i.e., (1) the single G^1 coaxial conic arc interpolator, and (2) the coaxial bi-conic interpolator. Globally, we fit the generatrix with *coaxial conic spline* (both single and bi-conic arc interpolator), by moving the coaxial conic arc interpolator adaptively and sequentially from one endpoint to the other one. Once a coaxial arc interpolator (both single and bi-conic arc interpolator) to fit the generatrix is determined by stretching it to the longest, we start the next *coaxial arc* fitting. This marching of coaxial arc interpola-

tor is repeated until the other endpoint of the generatrix is reached.

Coaxial conic arc fitting method consists of three major steps as follows:

- [Polygonization] To sample the generatrix into a set of points P_i and tangent vectors T_i ;
- [Local conic fitting] To construct a *coaxial conic interpolator* to fit the generatrix optimally (single or bi-conic arc interpolator);
- [Global conic fitting] To move *coaxial conic interpolator* along generatrix adaptively and progressively.

3.1. GC⁰ single coaxial conic fitting

It is worth noting that since our polygonization is independent of the form of the generatrix, it is applicable to both parametric and algebraic representations, and even the discrete form. Each conic arc C_j is defined on the interval $[x_m, x_n]$ ($0 \leq m \leq j \leq n \leq N$) and computed by using the least square method to estimate A_i , D_i and F_i . By minimizing the sum of square distance differences of all sampling points and two interpolation constraints at two endpoints $P_m(x_m, y_m)$ and $P_n(x_n, y_n)$, C^0 least square estimation of A_i , D_i and F_i is derived as follows:

$$f = \sum_{j=m+1}^{n-1} (Ax_j^2 + y_j^2 + Dx_j + F_i)^2 + \lambda_1 (Ax_m^2 + y_m^2 + Dx_m + F_i) + \lambda_2 (Ax_n^2 + y_n^2 + Dx_n + F_i)$$

to differentiate f with respect to A_i , D_i , F_i , λ_1 and λ_2 respectively,

$$f'_{A_i} = A_i \sum_{j=m+1}^{n-1} 2x_j^4 + \sum_{j=m+1}^{n-1} 2x_j^2 y_j^2 + D_i \sum_{j=m+1}^{n-1} 2x_j^3 + F_i \sum_{j=m+1}^{n-1} 2x_j^2 + \lambda_1 x_m^2 + \lambda_2 x_n^2 = 0,$$

$$f'_{D_i} = A_i \sum_{j=m+1}^{n-1} 2x_j^3 + \sum_{j=m+1}^{n-1} 2x_j y_j^2 + D_i \sum_{j=m+1}^{n-1} 2x_j^2 + F_i \sum_{j=m+1}^{n-1} 2x_j + \lambda_1 2x_m + \lambda_2 2x_n = 0,$$

$$f'_{F_i} = A_i \sum_{j=m+1}^{n-1} 2x_j^2 + \sum_{j=m+1}^{n-1} 2y_j^2 + D_i \sum_{j=m+1}^{n-1} 2x_j + 2(n-m-2)F_i + \lambda_1 + \lambda_2 = 0,$$

$$f'_{\lambda_1} = A_i x_m^2 + y_m^2 + D_i x_m + F_i = 0,$$

$$f'_{\lambda_2} = A_i x_n^2 + y_n^2 + D_i x_n + F_i = 0.$$

We can obtain A_i , D_i , and F_i , by solving the following linear equation system:

$$Hc = b, \quad (1)$$

Where

$$H = \begin{bmatrix} \sum 2x_j^4 & \sum 2x_j^3 & \sum 2x_j^2 & x_m^2 & x_n^2 \\ \sum 2x_j^3 & \sum 2x_j^2 & \sum 2x_j & x_m & x_n \\ \sum 2x_j^2 & \sum 2x_j & 2(n-m-2) & 1 & 1 \\ x_m^2 & x_m & 1 & 0 & 0 \\ x_n^2 & x_n & 1 & 0 & 0 \end{bmatrix}, \quad c = \begin{bmatrix} A_i \\ D_i \\ F_i \\ \lambda_1 \\ \lambda_2 \end{bmatrix}$$

$$c^T = [A_i, D_i, F_i, \lambda_1, \lambda_2],$$

$$b^T = [-\sum 2x_j^2 y_j^2, -\sum 2x_j y_j^2, -\sum 2y_j^2, -y_m^2, -y_n^2].$$

However, H is possibly singular if two given endpoints coincide, that implies, the determinant of H becomes zero if $x_m = x_n$. In this case, the equation (1) has no unique solution. We have two methods to solve this problem: (1) we can restrict each fitting curve segment to be an open one, because the proposed global coaxial conic fitting is just a marching procedure, the fitting interval can be controlled freely by shrinking or stretching it to avoid such closed loop during the conic fitting; (2) we can calculate the pseudo inverse H^+ of H [7], instead of the real inverse of H , that is a least square solution to H^+ .

3.2. GC¹ coaxial bi-conic splines

A single coaxial conic arc G^1 interpolating two given endpoints and two associate tangential vectors requires *four* constraints of two position interpolations and two tangential interpolations in total, but an axial conic arc has only three freedoms - A , D , and F - and is thus over-constrained for the four constraints. Here, instead of a *single* coaxial conic arc, we propose a *coaxial bi-conic arc spline* to G^1 interpolate two specified endpoints $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ and their associated tangent vectors $T_1(T_{x1}, T_{y1})$ and $T_2(T_{x2}, T_{y2})$, by using *two* coaxial conic arcs, which preserves continuity at their common endpoint $P_m(x_m, y_m)$. This idea is inspired by *circular biarc spline*. Two coaxial conic arcs $C_1: A_1x^2 + y^2 + D_1x + F_1 = 0$ on the interval $[x_1, x_m]$, and $C_2: A_2x^2 + y^2 + D_2x + F_2 = 0$ on the interval $[x_m, x_2]$ are used together to fit the generatrix on the interval $[x_1, x_2]$, which provide *six* freedoms $A_1, D_1, F_1, A_2, D_2, F_2$ to solve the six constraints (by G^1 position and tangential interpolation on P_1, P_2 and P_m) as follows:

- C_1 interpolates at the endpoint P_1 ;
- C_1 interpolates at tangent vector T_1 ;
- C_2 interpolates at the endpoint P_2 ;
- C_2 interpolates at tangent vector T_2 ;
- C_1 joins C_2 at $P_m(x_m, y_m)$ with G^0 continuity;
- C_1 joins C_2 at $P_m(x_m, y_m)$ with G^1 continuity.

Here, P_m is not asked definitely to lie on the generatrix in order to fit it more flexibly. Because y_m depends on x_m , there is only one constraint x_m actually. Six linear equations are formed for estimating six coefficients $A_1, D_1, F_1, A_2, D_2, F_2$ of *coaxial bi-conic arc interpolator* as follows:

$$\begin{cases} A_1 x_1^2 + y_1^2 + D_1 x_1 + F_1 = 0 \\ (2A_1 x_1 + D_1) T_{x1} = 2y_1 T_{y1} \\ A_2 x_2^2 + y_2^2 + D_2 x_2 + F_2 = 0 \\ (2A_2 x_2 + D_2) T_{x2} = 2y_2 T_{y2} \\ A_1 x_m^2 + y_m^2 + D_1 x_m + F_1 = A_2 x_m^2 + y_m^2 + D_2 x_m + F_2 \\ \frac{2A_1 x_m + D_1}{y_m} = \frac{2A_2 x_m + D_2}{y_m} \end{cases}$$

By eliminating y_m in the fifth and sixth equations, we obtain the system (2) of linear equations about x_m as

$$\begin{cases} A_1x_1^2 + y_1^2 + D_1x_1 + F_1 = 0 \\ (2A_1x_1 + D_1)T_{y1} = 2y_1T_{x1} \\ A_2x_2^2 + y_2^2 + D_2x_2 + F_2 = 0 \\ (2A_2x_2 + D_2)T_{y2} = 2y_2T_{x2} \\ A_1x_m^2 + Dx_m + F_1 = A_2x_m^2 + D_2x_m + F_2 \\ 2A_1x_m + D_1 = 2A_2x_m + D_2 \end{cases} \quad (2)$$

The coefficients $A_1, D_1, F_1, A_2, D_2,$ and F_2 can then be expressed explicitly as functions of the two end points and their associated tangents, in addition to x_m which is the only unknown variable that is left as a freedom for the optimal bi-conic fitting. Similar to circular biarc spline, *coaxial bi-conic arc spline* also has two types, (1) *C-type coaxial bi-conic arc spline* as shown in Fig. 2, it may be composed of two joining elliptic arcs; and (2) *S-type coaxial bi-conic arc spline* as shown in Fig. 3, it may be composed of an elliptic/parabolic arc joining with a coaxial hyperbolic arc together. Combining the two types of coaxial bi-conic splines can flexibly fit arbitrary generatrices. Thus, such coaxial bi-conic arc spline successfully resolves the over-constrained problem faced by the single coaxial conic fitting.

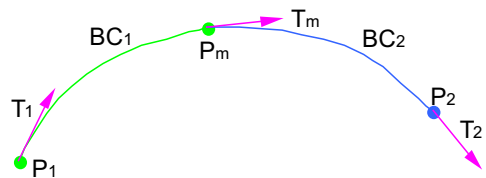


Figure 2. C-type coaxial bi-conic arc spline

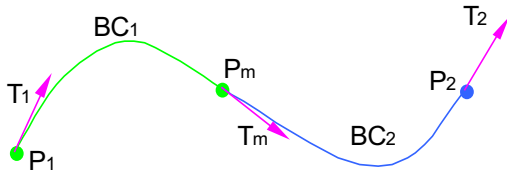


Figure 3. S-type coaxial bi-conic arc spline

3.3 Optimal coaxial bi-conic arc fitting

In the above equation system (2), x_m is an unknown variable, apparently, it is a freedom for optimally fitting the generatrix. As shown in Fig. 4, choosing different section point $x_m = x_1 + t(x_2 - x_1)$ ($0 \leq t \leq 1$) can produce different approximation effects, the problem is how to find the best t for the optimal coaxial bi-conic arc fitting. Theoretically, it can be transformed into a complicated optimization problem as follows:

$$e(x_m) = \left\{ \int_{x_1}^{x_m} |C_1(x, x_m) - G(x)|^2 dx + \int_{x_m}^{x_2} |C_2(x, x_m) - G(x)|^2 dx \right\} \quad (3)$$

where $C_1(x, x_m)$ and $C_2(x, x_m)$ are the explicit equations of the two coaxial bi-conic arcs, and $G(x)$ is the explicit representation of the generatrix.

The optimization problem (3) is rather complicated usually and cannot be solved easily. Considering that *golden section searching* is the simplest optimization method and easy to implement, we choose it to determine the best t_m . Its main idea is to compare the maximum errors e_l and e_2 at two golden points $t_m = t_0 + 0.382(t_1 - t_0)$ and $t_u = t_0 + 0.618(t_1 - t_0)$ within the interval $[0, 1]$ (initially, $t_0 = 0.0, t_1 = 1.0$), and see if $e_m > e_u$, then the subinterval $[t_0, t_m]$ is thrown away, otherwise the subinterval $[t_u, t_1]$ is thrown away. By performing such golden section searching recursively until $[t_m, t_u]$ is small enough (less than the specified threshold ε , ε is set as $e/10$ in our program, where e is the specified error tolerance), the best t_b is set as $(t_m + t_u)/2$, the mid-point of $[t_m, t_u]$. However, such searching may only find a locally optimal point and miss the global optimal t_m when the objective function is a multimodal one.

To avoid missing the global optimal point, in each round of compressing the interval $[t_m, t_u]$, we choose $e_m = \min(e_0, e_m)$ and $e_u = \min(e_2, e_u)$. If $e_m < e_0$, we set $t_0 = t_m, t_m = t_0 + 0.382(t_1 - t_0)$ and $e_0 = e_m$. Similarly, if $e_u < e_1$, we set $t_1 = t_u, t_u = t_0 + 0.618(t_1 - t_0)$ and $e_1 = e_u$. Then we estimate new e_m and e_u as to new t_m and t_u , and keep such compressing the interval $[t_m, t_u]$ until finding the best section point x_b .

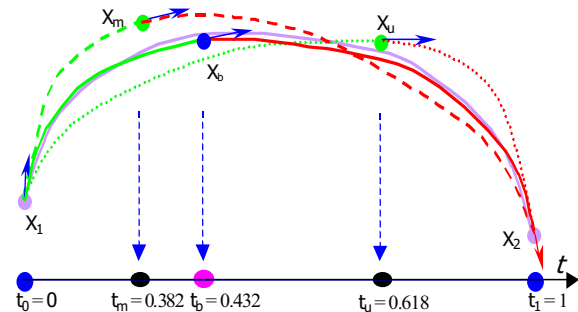


Figure 4. Finding the best x_m by golden ratio searching

3.4. Bisection marching

Under the specified tolerance d , how to stretch each fitting conic arc interpolator (both single and bi-conic) along the generatrix as long as possible for achieving the smallest number of fitting conic arcs globally? Adaptive error-driven *bisection marching* is employed in our method. Initially, the first conic arc interpolator is used to fit the generatrix on $[x_1, x_2]$, and then we compare the error measure e with d , where, e is measured by averaging the maximum horizontal and vertical distances between the original generatrix and the fitting conic arc interpolator. The global marching idea can be sketched as follows:

Repeat the following three steps until $x = x_2$:

- If $d = e$, end this round of fitting conic arc interpolator and start next round of conic arc interpolator fitting from x ;
- If $d < e$, enlarge x to $(x+x_2)/2$ and redo the conic fitting on the new interval $[x_1, (x+x_2)/2]$;
- If $d > e$, reduce x to $(x_1+x)/2$ and redo the conic fitting on the new interval $[(x+x_1)/2, x]$.

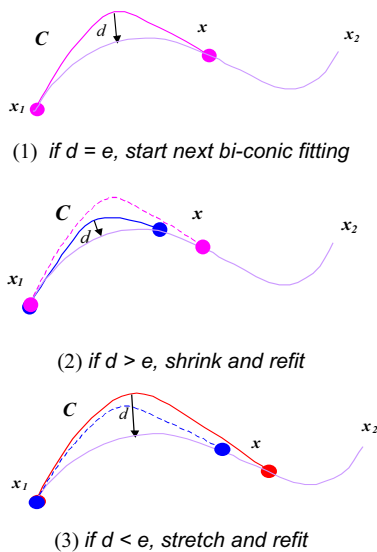


Figure 5. Bisection marching bi-conic arc interpolator

3.5 An experimental example

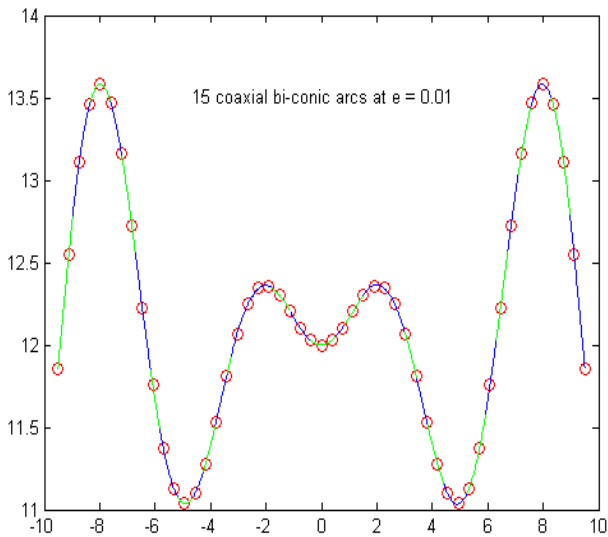


Figure 6. Coaxial bi-conic fitting generatrix with $e = 10^{-2}$

We implemented an example as shown in Fig. 6 by using linear approximation (*truncated cone decomposition*), G^0 single coaxial conic fitting and G^1 coaxial bi-conic arc fitting (*revolute quadric decomposition*) respectively. The performance data in Table 1 shows that the ratio of the number of the required linear segments over the number of

the required conic arcs is reciprocal to the error measure e . At the tolerance 10^{-1} , the number of line segments is 1.75 times of the number of the required conic arcs. At the tolerance 10^{-3} , the number of line segments is 3 times of the number of conic arcs. This ratio increases to 6.5 times when the tolerance is 10^{-5} . Therefore, the revolute quadric decomposition is considerably more efficient than the truncated cone *decomposition* for surfaces of revolution. The major computations are analytical, and the algorithm guarantees that coaxial bi-conic arcs interpolate at the sampling points and the associated tangent vectors on the generatrix exactly. It is robust, simple and easy to implement in practice. It should be pointed out that although the single-revolute-quadric-based decomposition generates less conic arcs than the bi-revolute-quadratics-based decomposition, it is at the cost of losing G^1 continuity on the approximated generatrix, which is usually prohibited in most applications.

Table 1. Performance data of three quadric decompositions, truncated cone (TC) vs. G^0 single coaxial conic fitting (single RQ) vs. G^1 coaxial bi-conic fitting (Bi-RQ)

| Tolerance | | 10^{-2} | 10^{-3} | 10^{-4} | 10^{-5} | 10^{-6} | 10^{-7} |
|----------------------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| Num. of fitting conic arcs | TC | 42 | 134 | 421 | 1329 | 4202 | 13284 |
| | Single RQ | 18 | 34 | 70 | 152 | 325 | 689 |
| | Bi-RQ | 30 | 52 | 104 | 212 | 536 | 910 |

4. Intersection determination

This section addresses how to take fast intersection detections. Firstly, *cylindrical bounding shell* (CBS) is devised for fast intersection rejection of plane and surfaces of revolution. Although a surface of revolution is subdivided into many trimming coaxial revolute quadrics, not all of them have intersection with the cutting plane, usually, only very few of them have real intersection with the surface of revolution. Then, a notation, *valid intersection intervals* (VII), is introduced to rule out those revolute quadrics that are guaranteed not to intersect the cutting plane, for which it is unnecessary to perform intersection computations. Furthermore, *cylindrical bounding shell clipping technique* is proposed to efficiently determine VII of a plane intersecting a surface of revolution.

4.1. Cylindrical bounding shell (CBS)

Bounding volume is an important technique for efficiently computing surface intersections. It can help do easy and fast intersection rejections. Two popular types of bounding volumes are bounding sphere and bounding box; [3] and [11] propose using bounding cylinder specifically

for surfaces of revolution. In our method, we use cylindrical bounding shell (CBS) (see Fig. 7(a)) to enclose a surface of revolution both internally and externally by using the double bounding cylinders, with the maximum and minimum distances of the generatrix to the axis as the internal and external radii respectively. It is based on two considerations:

- It can enclose a surface of revolution more tightly than the traditional bounding box, bounding sphere, or a single bounding cylinder;
- Intersection determination of plane/cylinder (geometric method) is much easier to do than intersection determination of plane/revolute quadric, which can be further reduced to intersection determination of line/rectangle by projection along the cross product of the axial vector and normal vector of the cutting plane, which can be done by using only linear computations.

4.2. Valid intersection intervals (VII)

Valid intersection interval (VII) is defined as a set of axial Z-intervals, used to find out all the revolute quadrics for valid intersection computations. We only need to take intersection tests for these revolute quadrics. Cylindrical bounding shell and valid intersection intervals can combine together to make rough intersection detection as follows:

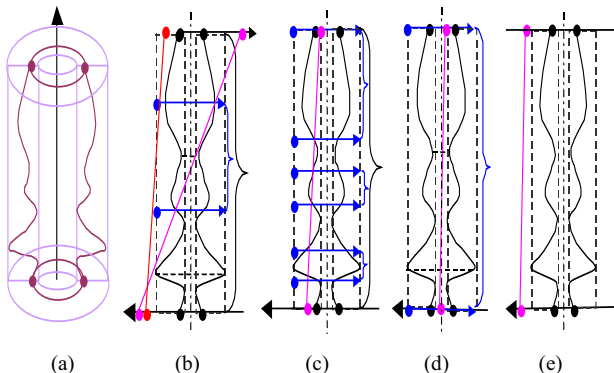


Fig. 7 Valid intersection intervals VII and cylindrical bounding shell of surfaces of revolution

- If a plane intersects the exterior bounding cylinder, then it may either intersect the surface of revolution or not, e.g. in Fig. 7(b), the red plane does not intersect the surface of revolution, but the pink plane traverses the surface of revolution horizontally and produces a single VII (blue subintervals);
- Plane cuts both bottom or top circles and produces multiple VII, as shown in Fig. 7(c);
- If a plane intersects the interior bounding cylinder, then it definitely intersects the surface of revolution. If the plane cuts the interior bounding cylinder at the top and bottom circles vertically as shown in Fig. 7(d),

then the plane intersects all the subdivided RQs and VII contains all the Z-interval of generatrix;

- If a plane has no intersection with the exterior bounding cylinder as shown in Fig. 7(e), then the plane is impossible to intersect the surface of revolution and VII becomes an empty set.

4.3. Cylindrical bounding shell clipping

Determining VII plays an important role on efficient computing of plane sections of surfaces of revolution. However, computing VII exactly and directly is a non-trivial task due to the need of solving a high-degree polynomial equation with numerical solutions. Here, we propose a simple algorithm to approximate VII by refining the potential intersection intervals of plane and CBS recursively, it is noted as *cylindrical bounding shell clipping*; as a result, it becomes much easier to compute the VII of the plane and the bounding cylinder. A plane clipping a CBS may form three types of intersection intervals as follows:

- Definite intersection intervals V_{in}
 - The plane clips the interior bounding cylinder.
 - The revolute quadrics within V_{in} intersect the plane definitely.
 - Put it into VII.
- Null intersection intervals V_{ex}
 - The plane clips the exterior bounding cylinder.
 - The revolute quadrics within V_{ex} definitely do not intersect the plane.
- Potential intersection intervals V_{po} between V_{ex} and V_{in}
 - The revolute quadrics within V_{po} may or may not intersect the cutting plane.
 - It still needs further clipping CBS (recursive refinement).

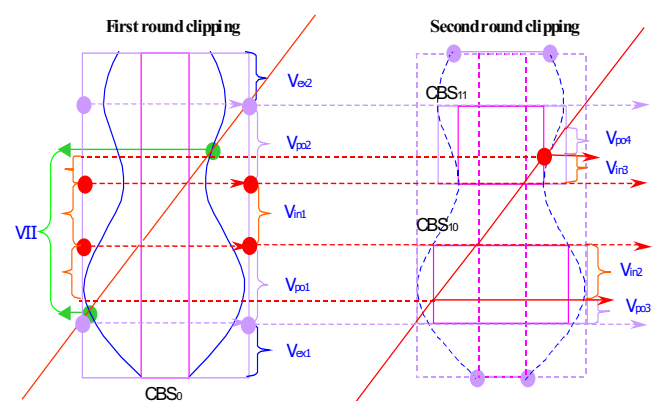


Figure 8. Two consecutive cylindrical bounding shell clippings for intersection searching

Each clipping of CBS produces three kinds of intersection intervals by the plane clipping interior and exterior bounding cylinders respectively. The first round of clipping of the initial CBS CBS_0 rules out null intersection intervals V_{ex1} and V_{ex2} , merges definite intersection interval V_{in} into VII, and produces potential intersection intervals V_{po1} and V_{po2} . In the second round of clipping, we construct two new cylindrical bounding shells CBS_{10} and CBS_{11} for those revolute quadrics within two potential intersection intervals V_{po1} and V_{po2} and clip them respectively. Plane clipping CBS_{10} produces a definite intersection interval V_{in2} and a potential intersection interval V_{po3} . Plane clipping CBS_{11} produces a definite intersection interval V_{in3} and a potential intersection interval V_{po4} . Put two new definite intersection intervals V_{in2} and V_{in3} into VII. After the second CBS clipping, the union of three definite intersection intervals V_{in1} , V_{in2} and V_{in3} has approximated the real VII very well. The third round of CBS clipping can be done on the two new potential intersection intervals V_{po3} and V_{po4} recursively. So CBS clipping technique is a good way to estimate VII. The following *algorithm1* sketches the above idea. We can end the recursive procedure of refinement if (i) the potential intervals cannot be shortened in the two consecutive refinements, or (ii) the current potential (purple) intersection intervals only contain less than 3 revolute quadrics.

Algorithm 1. Computing the valid intersection intervals (VII) of a plane and a surface of revolution

Proc *Search_PR_VII* (plane, CBS_0)

Input: plane, CBS_0 ;

Output: valid intersection intervals VII (initially, it is set as nil);

Begin Proc

```

1  {  $I_{exterior}$ ,  $I_{interior}$ ,  $I_{potential}$  } = PlaneClipCBS(plane,  $CBS_0$ )
2  if  $I_{interior}$  is not empty
3    VII = VII + {  $I_{interior}$  };
4  end if;
5  if  $I_{potential}$  contains less than 2 RQs
6    VII = VII + {  $I_{potential}$  };
7    return VII;
8  else
9    locate all the RQs within  $I_{potential}$ ;
10   if the number of RQs within  $I_{potential}$  decreases
11     form new cylindrical bounding shells  $CBS_{10}$  and
            $CBS_{11}$ ;
12     Search_PR_VII(plane,  $CBS_{10}$ );
13     Search_PR_VII(plane,  $CBS_{11}$ );
14   else // CBS clipping cannot shrink  $I_{potential}$ 
15     return VII;
16   end if;
17 end if;

```

End Proc.

5. Computing and tracing plane sections of surfaces of revolution

Basically, computing the intersection curves consists of three integral parts, (1) computing the intersection curves efficiently, (2) classifying the intersection curves topologically, and (3) representing the intersection curves rationally. Because our idea is to subdivide a surface of revolution into a set of \mathcal{G}^1 coaxial revolute quadrics along its generatrix, the intersection problem of surfaces of revolution is reduced to the intersection problem of revolute quadrics. For (1), we propose CBS clipping for VII and employ the fundamental geometric method in [19] to compute the plane sections of revolute quadric. For (2), we propose a simple algorithm of tracing the final intersection curves topologically as a group of independent branches and loops. For (3), the final intersection curves are composed of a group of \mathcal{G}^1 conic arcs, which can be represented naturally as piecewise rational Bezier curves or NURBS curves, two standard forms in CAD/CAM systems. Detailed derivation can be found in [15].

How to detect the connected components and singular points of the intersection curves are important issues of boundary representation in solid modeling. This problem becomes relatively easier when tracing the intersection curves of plane and surfaces of revolution - it becomes how to connect those conic arcs into the closed loops and recognize the singular points during tracing. Here, we present a simple algorithm of tracing the intersection curves of a plane and a surface of revolution. Basically, there are three types of the trimming intersection curves (conic arcs) of a plane and a revolute quadric:

- *Isolated point.* It is a tangential point of the plane and a revolute quadric, in this case, the algorithm reports this point as a singular point.
- *Double branch.* In this case, the plane has two intersection points with each bounding circle respectively (top and bottom) for a truncated revolute quadric, the resulting intersection curve becomes a pair of symmetric conic arcs (elliptic, hyperbolic, parabolic, two separated line segments, two crossing line segments). We concatenate them at their coincident endpoints along the generatrix sequentially.
- *Single branch.* There are four cases: (i) the plane has intersection points with only one bounding circle (top or bottom circle), the resulting intersection curve is a trimming conic arc; (ii) the plane has intersection points with neither of the two bounding circles, the resulting intersection curve is a complete conic section, which itself is a closed loop; (iii), the plane has two intersection points with one bounding circle and one intersection (tangential) point with the other bounding circle. Such single branch begins or ends a closed loop

during tracing; finally, (iv) when a revolute quadric is a truncated cone, its planar section is a single line segment, which is a degenerated case of a double-branch and can be treated as a double-branch during tracing.

The following algorithm outlines the rough idea of tracing these conic arcs piece by piece along the spine curve sequentially, by checking coincidence of endpoints of two adjacent conic arcs, and then concatenating them into a set of connected components one by one.

Algorithm 2. Tracing plane sections of surface of revolution

Proc. *Tracing_Sect_RS(plane,RS)*

Input: a list of conic arcs $CS_i (i=1,2, \dots, n)$;

Output: a list of branches $PB, (PB_j, j=1,2, \dots, m)$;

Begin Proc

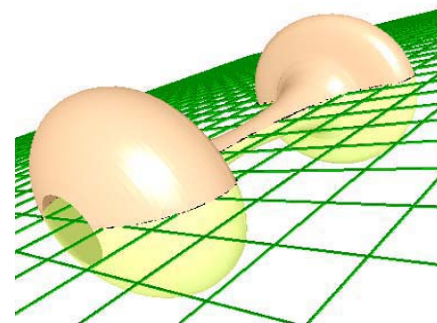
1. $j = 1$;
2. **for** $i = 1$ to m
3. **if** PB_j is empty
4. **insert** CS_i into the current component PB_j ;
5. **if** CS_i is a complete conic section
6. **close** the current component PC_j ;
7. $j++$;
8. **end if**;
9. **if** CS_i is an isolated point
10. **close** the current component PC_j ;
11. $j++$;
12. **identify** it as the singular point;
13. **end if**;
14. **else**
15. **if** CS_i is a single or double-touch to current PC_j
16. **concatenate** it into the current branch PC_j ;
17. **close** the current branch PC_j ;
18. $j++$;
19. **else**
20. **concatenate** it into the current branch PC_j ;
21. **end if**;
22. **end if**;
23. **end for**;

End Proc.

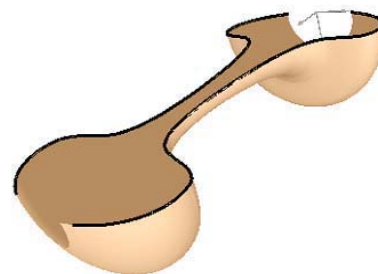
6. Implemented examples

We have implemented the presented revolute quadric decomposition idea and the intersection algorithm and three examples are given here to testify the robustness and efficiency of our algorithm. The first example is a pair of symmetric open branches by a plane intersecting a dumb-bell as shown in Fig. 9. The second example is two closed loops by a plane intersecting a bottle as shown in Fig. 10. The third example is three branches by a plane intersecting a chalice as shown in Fig. 11, among them, two are closed

loops and one is an open branch. For comparison purpose, we also implemented plane sections of surface of revolution based on truncated cone decomposition.

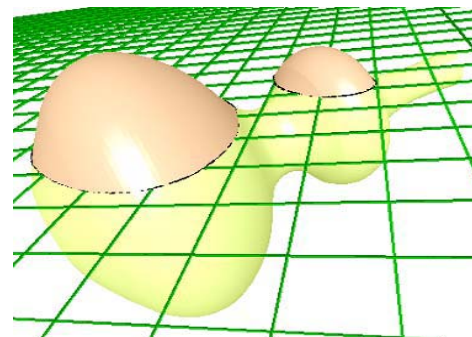


(1)

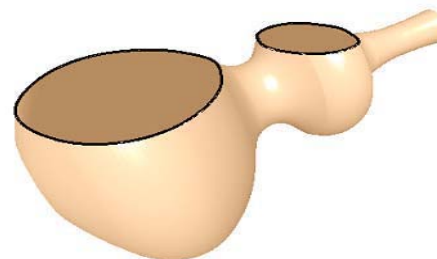


(2)

Figure 9. The Intersection curve of two open branches

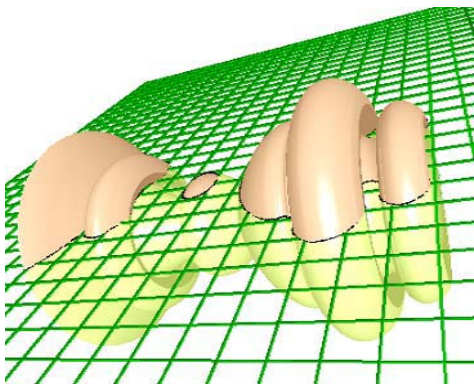


(1)

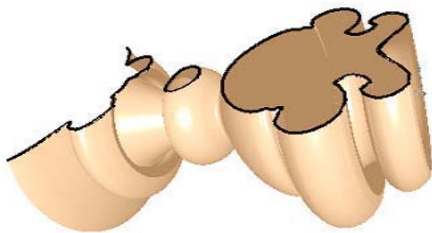


(2)

Figure 10. The two-loop intersection curves



(1)



(2)

Figure 11. The intersection curves with one open branch and two loops

Table 2. Comparison between truncated cone and revolute quadric decomposition at tolerance 10^{-4}

| Examples | Truncated cone decomposition | | Revolute quadric decomposition | |
|----------|------------------------------|--------------------|--------------------------------|--------------------|
| | Consumed memory | Computing time (s) | Consumed memory | Computing time (s) |
| Dumbbell | 420 KB | 0.107s | 63.7 KB | 0.039s |
| Bottle | 436 KB | 0.125s | 72.2 KB | 0.059s |
| Chalice | 1.2 MB | 0.455s | 258KB | 0.23s |

Table 3. Comparison between the intersection algorithms based on truncated cone and revolute quadric decomposition at the tolerance 10^{-4}

| Examples | Truncated cone decomposition | | Revolute quadric decomposition | |
|----------|------------------------------|-------------------|--------------------------------|-------------------|
| | # of fitting conic arcs | Intersection time | # of fitting conic arcs | Intersection time |
| Dumbbell | 41 | 0.00032s | 14 | 0.000014s |
| Bottle | (13, 9) | 0.005s | (4, 4) | 0.00062s |

| | | | | |
|---------|--------------|-------|-------------|-------|
| chalice | (66, 10, 27) | 0.04s | (27, 6, 13) | 0.01s |
|---------|--------------|-------|-------------|-------|

From the performance data given in Table 2, the revolute quadric decomposition based algorithm outperforms the truncated cone decomposition based algorithm in terms of both the CPU time and consumed memory. The tighter the tolerance, the larger difference in both computing time and consumed memory. Also, Table 3 shows that, in the computing time and total number of conic arcs of the intersection curves of a plane and a surface of revolution, the revolute quadric decomposition based algorithm outperforms greatly the truncated cone decomposition based algorithm. Assuming that the total fitting components is N , both the computing time and memory cost for decompositions are $O(N)$. But, for intersections, the computing time and memory cost is only $O(\log N)$ on average, with the help from CBS clipping. It is noted that the decomposition time is much more than the intersection time; however, since the decomposition belongs to preprocessing stage, it is executed only once, while the intersection operations are performed frequently and recurrently.

7. Conclusion

Quadric decomposition brings a new perspective to solving surface intersection problem. It reduces surface intersection problem to quadric intersection problems. In this paper, we revisit the problem of plane intersecting surfaces of revolution by proposing using revolute quadric decomposition for approximating surface of revolution. Implementation results show that the revolute quadric decomposition outperforms the truncated cone decomposition greatly in terms of the computing time, consumed memory, and total number of the required conic arcs. It also is more convenient to represent the intersection curves rationally and much easier to trace the intersection curves for identifying the closed loops and singular points than some existing methods (e.g., the Kim's method [13]). Since we employ a fundamental geometric method [19] to compute plane sections of revolute quadrics which is robust, efficient and accurate, so is our algorithm. Its accuracy can be specified by the precision of quadric subdivision of surface of revolution in advance. The other contributions are cylindrical bounding shell and plane clipping CBS technique which are proposed for valid intersection interval detection.

The presented method of globally fitting a surface of revolution by revolute quadrics is just a sub-optimal method, developing a true optimal method becomes our future work. Quadric decomposition also can be applied to computing the intersection problem of two surfaces of revolution and other geometric computing problems of surfaces of revolution, e.g. isophot, bisector, distance computing and so on.

8. References

- [1] Baciú G., Kwok K. W. and Jia J. Y., "An efficient method of computing planar sections of surfaces of revolution", *Proceedings of CAD & CG*, pp. 111-119, Academic Publisher, Kunming, China, 2001.
- [2] Bangert, C. and Prautzsch, H., "Quadric spline", *CAGD*, 16(6), pp. 497-515, 1999.
- [3] Burger P. and Gillies D., "Rapid Ray Tracing of General Surfaces of Revolution", *New Advances in Computer Graphics, Proceedings of CGI'89*, Springer-Verlag Press, 1989.
- [4] Dahmen W., "Smooth piecewise quadric surfaces", *Mathematical methods in computer aided geometric design*, edited by Lyche T. and Schumaker, L., Academic Press, pp. 181-194, 1989.
- [5] Froumentin M. and Chaillou C., "Quadric surfaces: a survey with new results", *The Mathematics of Surfaces VII*, pp. 363-381, Edited by Tim Goodman and Ralph Martin, 1997.
- [6] FU Q. X., "The intersection of a bicubic Bezier patch and a plane", *Computer Aided Geometric Design*, 7(6), pp. 475-488, 1990.
- [7] Golub G. H., Loan C. F., "Matrix computations", Johns Hopkins University Press, Edition 3, Baltimore, 1996.
- [8] Goldman R. N., "Intersection of parametric surface and a plane", *IEEE Computer Graphics & Applications*, 4 (8), pp. 48-51, 1984.
- [9] Guo B., "Quadric and cubic bitetrahedral patches", *The Visual Computer*, Vol. 7, pp. 253-262, 1991.
- [10] Guo B., "Representation of arbitrary shapes using implicit quadrics", *The Visual Computer*, Vol. 9, pp. 267-277, 1993.
- [11] Jia J., "Rapid Ray Tracing Surfaces of Revolution", *Master Thesis of Jilin University*, China, 1991.
- [12] Johnstone J. K. and Shene C. K., "Computing the intersection of a plane and a natural quadric", *Computers & Graphics*, 16 (2), pp. 179-186, 1992.
- [13] Kim K. J., Kim M. S., and Martin R., "Intersecting a translationally or rotationally swept surface with a plane of a sphere", *Proceeding of Korea-Israel Bi-National Conference on Geometric Modeling and Computer Graphics*, pp. 165-193, Seoul, Korea, September 30 - October 1, 1999.
- [14] Kim M. S., "Intersecting surfaces of special types", *Proceeding of Shape Modeling and Processing 99*, pp.122-128, Univ. of Aizu, Japan, March 1999.
- [15] Kwok K. W., "Efficient computing intersection curves of surfaces of revolution", Master of Philosophy Thesis, Department of Computer Science, HKUST, May, 2001.
- [16] Miller J. R. and Goldman R. N., "Using tangent ball to find plane sections of natural quadrics", *IEEE Computer Graphics & Applications*, 12(2), pp 68-82, 1992.
- [17] Patrikalakis N. and Maekawa T., "Intersection Problem", *Handbook of Computer Aided Geometric Design*, Farin G., Hoschek J. and Kim M. S. (Eds.), Elsevier, Amsterdam, 2001.
- [18] Powell, M. J. D. and Sabin M. A., "Piecewise quadratic approximations on triangles", *ACM Transaction on Mathematical Software*, Vol. 3, pp. 316-325, 1977
- [19] Shene C. K. and Johnstone J. K., "Computing the intersection of a plane and a revolute quadric", *Computers & Graphics*, 18 (1), pp. 47-60, 1994.
- [20] Zheng J. L. and Millham C. B., "A linear pivoting method for detecting and tracing planar section curves of free form surfaces", *Computer & Graphics*, 16(4), pp. 411-420, 1992.