Polynomial/Rational Approximation of Minkowski Sum Boundary Curves*

In-Kwon Lee*, Myung-Soo Kim*, and Gershon Elber+
* Department of Computer Science, POSTECH, Pohang 790-784, South Korea
+ Department of Computer Science, Technion, IIT, Haifa 32000, Israel

Abstract

Given two planar curves, their convolution curve is defined as the set of all vector sums generated by all pairs of curve points which have the same curve normal direction. The Minkowski sum of two planar objects is closely related to the convolution curve of the two object boundary curves. That is, the convolution curve is a superset of the Minkowski sum boundary. By eliminating all redundant parts in the convolution curve, one can generate the Minkowski sum boundary. The Minkowski sum can be used in various important geometric computations, especially for collision detection among planar curved objects.

Unfortunately, the convolution curve of two rational curves is not rational, in general. Therefore, in practice, one needs to approximate the convolution curves with polynomial/rational curves. Conventional approximation methods of convolution curves typically use piecewise linear approximations, which is not acceptable in many CAD systems due to data proliferation. In this paper, we generalize conventional approximation techniques of offset curves and develop several new methods for approximating convolution curves. Moreover, we introduce efficient methods to estimate the error in convolution curve approximation. This paper also discusses various other important issues in the boundary construction of the Minkowski sum.

Keywords: Convolution curve, offset curve, Minkowski sum, C-space obstacle, sweeping, curve approximation, Bézier curve, B-spline curve

*The research was supported in part by the Korean Ministry of Science and Technology under Grants 96-NS-01-05-A-02-A, and 96-NS-01-05-A-02-B of STEP 2000, and by KOSEF (Korea Science and Engineering Foundation) under Grant 96-0100-01-01-2.

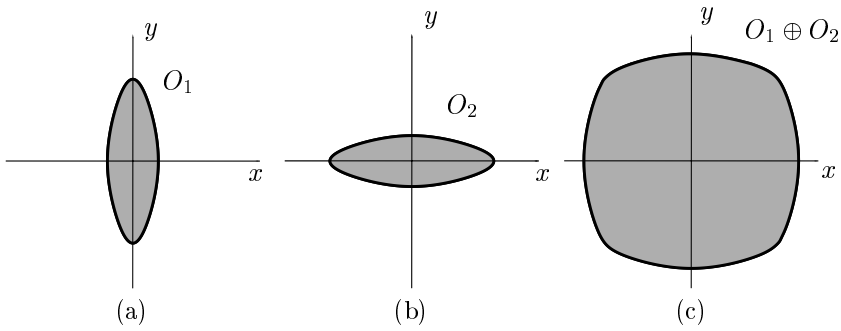


Figure 1: Minkowski sum $O_1 \oplus O_2$ of two planar convex objects O_1 and O_2.

1 Introduction

Convolution is a classic operation which has been used as a tool for computing collision–free paths in robot motion planning [3, 19, 27, 33]. Convolution is closely related to the notion of *Minkowski sum*. Given two planar curved objects O_1 and O_2, their Minkowski sum $O_1 \oplus O_2$ is defined as the set of all vector sums generated by all pairs of points in O_1 and O_2, respectively:

$$O_1 \oplus O_2 = \{a + b \mid a \in O_1,\ b \in O_2\}. \tag{1}$$

In Figure 1, the gray area of each object represents the object interior. The Minkowski sum of two planar objects considers all points in the interiors as well as on the boundaries of the two objects. In this paper, for the sake of computational efficiency and representational compactness, we are more concerned with constructing the boundary of the Minkowski sum.

Let O_1 and O_2 be two planar curved objects which are bounded by the planar curves C_1 and C_2, respectively. The problem of computing the Minkowski sum boundary, denoted as $\partial(O_1 \oplus O_2)$, can be transformed into the problem of computing the curve convolution of C_1 and C_2, denoted as $C_1 * C_2$ [3]. In the convolution operation, the vector sums are applied only to the pairs of curve points that have the same curve normal direction:

Definition 1.1 *Let $C_1(t) = (x_1(t), y_1(t))$ and $C_2(s) = (x_2(s), y_2(s))$ be two planar regular parametric curves. The convolution curve $C_1 * C_2$ is defined by*

$$(C_1 * C_2)(t) = C_1(t) + C_2(s(t)), \tag{2}$$

where

$$C_1'(t) \parallel C_2'(s(t)) \tag{3}$$

and

$$\langle C_1'(t), C_2'(s(t)) \rangle > 0, \tag{4}$$

for a reparametrization $s = s(t)$. □

When both O_1 and O_2 are convex objects, the convolution curve $C_1 * C_2$ is exactly the same as the Minkowski sum boundary $\partial(O_1 \oplus O_2)$ (see Figure 1). However, $\partial(O_1 \oplus O_2)$ is a subset of $C_1 * C_2$, in general [3]. Thus, the convolution curve $C_1 * C_2$ may have some redundant parts which do not contribute to the Minkowski sum boundary (see Figures 2(c) and 2(d)). To construct $\partial(O_1 \oplus O_2)$, we need to follow two steps: (i) compute the convolution curve $C_1 * C_2$, and (ii) eliminate the redundant parts of $C_1 * C_2$ which do not contribute to $\partial(O_1 \oplus O_2)$.

The convolution curve $C_1 * C_2$ is an envelope curve which is obtained by sweeping one curve C_1 (with a fixed orientation) along the other curve C_2 [3]. The constant radius offset curve is a special

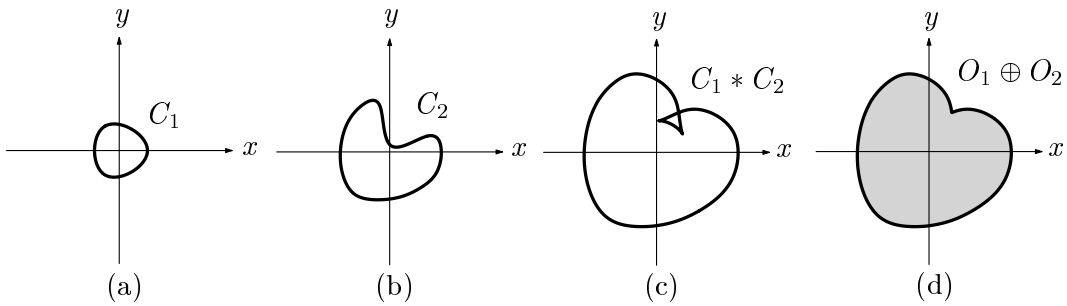


Figure 2: Minkowski sum $O_1 \oplus O_2$ of two planar objects O_1 and O_2: (a) convex object O_1 bounded by C_1, (b) non-convex object O_2 bounded by C_2, (c) convolution curve $C_1 * C_2$, and (d) the Minkowski sum $O_1 \oplus O_2$.

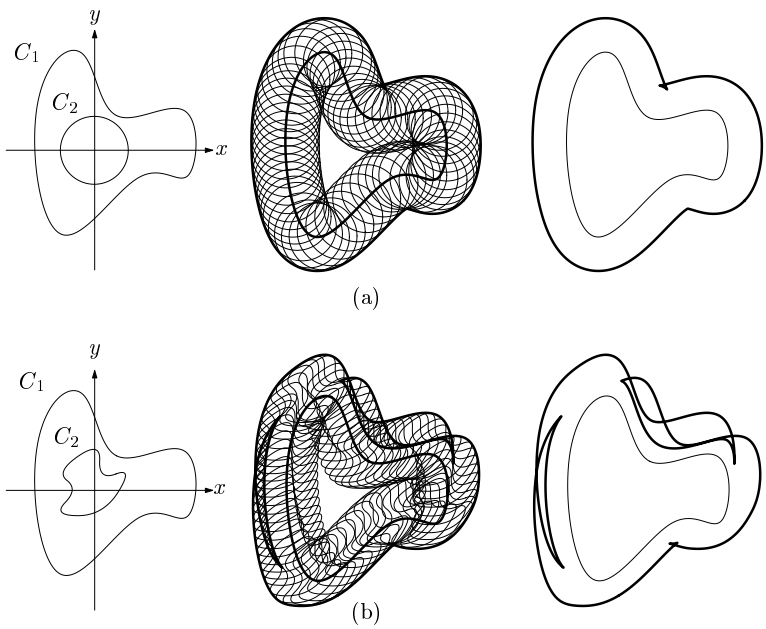


Figure 3: Offset and convolution curves: (a) offset, and (b) convolution

case of convolution curve in which the swept curve is restricted to being a circle of a fixed radius r. Figure 3 shows the offset and convolution curves of planar curves. In Figure 3(a), an offset curve is obtained by sweeping a circle C_2 along the other curve C_1 and taking only the outer envelope curve. In Figure 3(b), a convolution curve is generated by sweeping one curve C_2 along the other curve C_1 and similarly taking the outer envelope curve.

The Minkowski sum has been used as an important tool for computing collision–free paths in robot motion planning [3, 19, 27, 33]. Figure 4(a) shows the Minkowski sum $O_1 \oplus O_2$ of two planar curved objects O_1 and O_2. In Figure 4(b), we compute the Minkowski sum $O_1 \oplus (-O_2)$, where $-O_2$ is the symmetric object of O_2 with respect to the local reference point (which is located at the origin). There is no collision between O_1 and O_2 as long as the reference point of O_2 does not penetrate $\partial(O_1 \oplus (-O_2))$ (see Figure 4(c)). The object $O_1 \oplus (-O_2)$ is called the *Configuration-space (C-space) obstacle* of O_1 with respect to the moving object O_2.

The Minkowski sum has many other applications. In Figure 5, an outline font is designed by

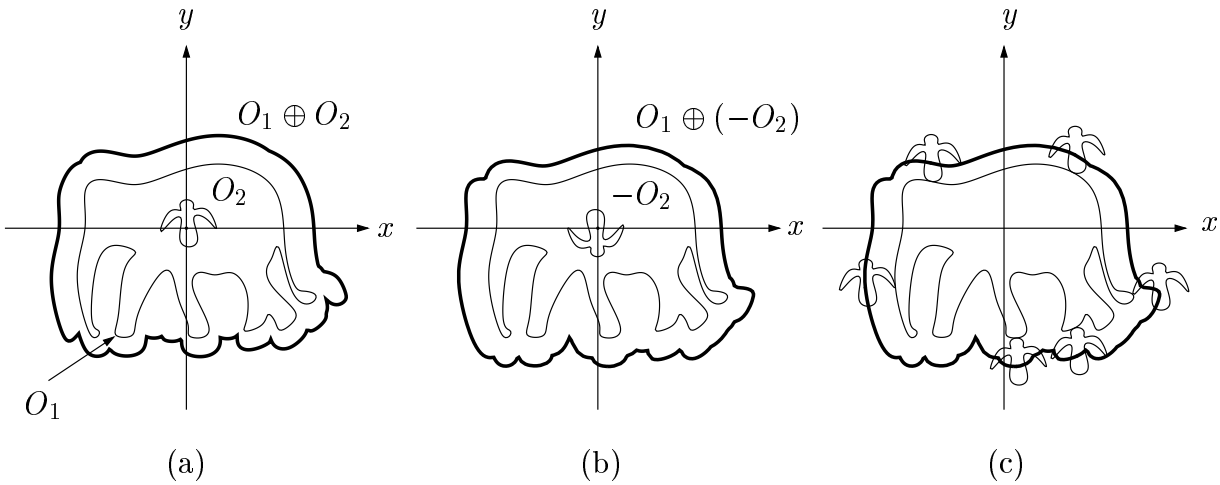


Figure 4: Minkowski sum and *C-space* obstacle

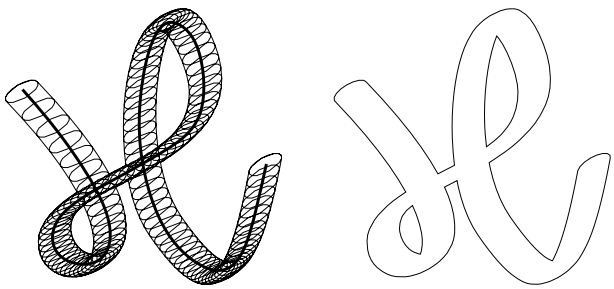


Figure 5: Outline font generation using the Minkowski sum

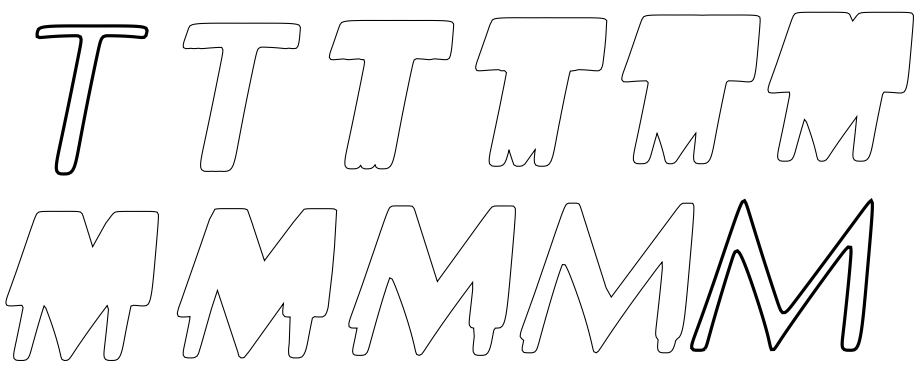


Figure 6: Shape transformation by a sequence of Minkowski sums

sweeping an ellipse (with a fixed orientation) along a skeleton curve [1, 15, 16]. The Minkowski sum can be used in shape transformation (i.e., metamorphosis or morphing) between two objects [24]. Figure 6 shows such an example of shape transformation between two planar objects. The intermediate shapes are the Minkowski sums of the character shapes 'T' and 'M' while scaling the T shape from 100 % to 0 % and the M shape from 0 % to 100 %, simultaneously.

Most CAD/CAM systems today represent curves and surfaces in polynomial/rational spline forms such as Bézier or NURBS (Non-Uniform Rational B-spline) curves and surfaces. Polynomial/rational curves and surfaces have many advantages (in rendering and geometric computations) over other representation methods such as implicit curves and surfaces [11]. Therefore, many conventional geometric modeling operations have been designed to deal with polynomial/rational curves and surfaces.

Given two planar algebraic curves, their exact convolution curve is also algebraic. Unfortunately, the convolution curve is not polynomial/rational, in general. Moreover, the convolution curve has a high algebraic degree. For example, the exact offset (a special case of the convolution) of a cubic Bézier curve has an algebraic degree of 10 [14]. These undesirable properties have led offset and convolution research to develop various approximation techniques that generate low degree polynomial/rational curves [10].

Consider Equation (3) which can be rewritten as follows:

$$y_2'(s(t))x_1'(t) - x_2'(s(t))y_1'(t) = 0. \tag{5}$$

When the curve $C_2(s)$ is a quadratic polynomial curve:

$$C_2(s) = (x_2(s), y_2(s)) = (as^2 + bs + c, ds^2 + es + f),$$

Equation (5) becomes

$$(2ds(t) + e)x_1'(t) - (2as(t) + b)y_1'(t) = 0,$$

which produces a unique solution for $s(t)$; namely, we have

$$s(t) = \frac{by_1'(t) - ex_1'(t)}{2dx_1'(t) - 2ay_1'(t)}.$$

When the curve $C_1(t)$ is rational, the convolution curve $(C_1 * C_2)(t)$ can be computed as a rational curve using a symbolic composition technique [8, 11, 13]. However, Equation (5) may not have a unique rational solution of $s(t)$ when the curve $C_2(s)$ has degree higher than two.

A simple way of approximating the convolution curve $(C_1 * C_2)(t)$ with a rational curve is to approximate: either (i) the curve $C_2(s)$ by a quadratic polynomial curve [30, 31], or (ii) the reparametrization $s(t)$ by a rational function [31]. Lee et al. [30] applied the first approach to the planar curve offset problem in which the curve $C_2(s)$ is an exact circle. Note that the circle can be represented exactly by a rational quadratic curve; but no polynomial curve can represent an exact circle. Therefore, Lee et al. [30] approximated the offset circle by a sequence of quadratic polynomial Bézier curves and approximated the offset curve by computing the convolution curves of the given base curve and the Bézier curves approximating the offset circle. Lee et al. [31] suggested convolution curve approximation methods based on the two approaches (i) and (ii) above, and compared their experimental results.

In Section 4, the methods of Lee et al. [31] are classified according to approaches based on: (i) quadratic curve approximation, (ii) reparametrization, and (iii) tangent field matching. Section 4 also presents other methods corresponding to approaches based on: (iv) control polygon and (v) interpolation. Elber et al. [10] surveyed conventional offset curve approximation methods and made detailed (qualitative and quantitative) comparisons of them. The approaches of Section 4 are generalizations of similar approaches in the offset curve approximation methods surveyed and compared in Elber et al. [10].

Conventional convolution curve approximation methods use piecewise linear approximations to represent the convolution curves (see Section 2), which may not be acceptable in many CAD systems due to data proliferation. Based on generalizations of conventional offset curve approximation methods, all the convolution curve approximation methods presented in this paper approximate the

convolution curves with polynomial/rational curves. Each of our methods also provides appropriate error analysis mechanism(s). (No previous method suggested a way to measure approximation error.)

This paper also discusses various other important issues in the construction of the Minkowski sum boundary; e.g., a compatible subdivision of input curves in a preprocessing step, the approximation of convolution curves with polynomial/rational curves, and the extraction of the Minkowski sum boundary from the planar graph of convolution curves. For the elimination of redundant parts in untrimmed convolution curves, we demonstrate a method based on a plane sweep algorithm [34] and apply the algorithm to piecewise linear approximations of the convolution curves. (There is no known implemented algorithm which can determine the arrangement of planar curve segments robustly; therefore, we use a robust algorithm that can determine the arrangement of approximating line segments.) Experimental results of this new trimming algorithm are promising.

The rest of this paper is organized as follows. In Section 2, we review previous work for the Minkowski sum and convolution curve computation. Section 3 describes a general algorithm for constructing the Minkowski sum boundary. In Section 4, we present several new methods that approximate convolution curves with polynomial/rational curves. Detailed qualitative and quantitative comparisons of these approximation methods are made in Section 5. In Section 6, we consider how to eliminate redundant parts of convolution curves for the construction of the Minkowski sum boundary. Some interesting experimental results are also demonstrated in this section. Finally, in Section 7, we conclude this paper and suggest further research problems. The implementation of all algorithms and comparisons presented in this paper is based on the IRIT solid modeling library [9].

2 Previous Work

In spite of the paramount importance of the Minkowski sum operation in practice, conventional convolution curve computation methods have many limitations. Exact methods [3, 15, 27] generate convolution curves which are algebraic/analytic curves; however, they are not rational, in general. Approximation methods [1, 25, 29] generate polygonal approximations, which may not be acceptable in many CAD systems due to data proliferation. Moreover, error analysis has not been seriously considered in the conventional approximation methods. The Minkowski sum boundary construction requires an algorithm which can determine the global arrangement of convolution curve segments in the plane; after that, it must eliminate all the redundant parts which lie in the Minkowski sum interior. Unfortunately, there has been no known implemented algorithm that can determine the curve arrangement in a robust way. A reasonable approximate solution may be based on using polygonal approximation of the convolution curves. These important issues have not been thoroughly considered in the previous work. We present more details in the following subsections. The later sections of this paper suggest some (partial) solutions to remedy drawbacks of the previous work.

2.1 The Minkowski Sum Computation

Lozano-Pérez [32, 33] used the Minkowski sum operation to construct the C-space obstacles for polygonal/polyhedral objects. To simplify the computation, each non-convex object is subdivided into convex objects and a convex Minkowski sum is computed for each pair of convex objects. After that, the Minkowski sum of the original objects is represented as a union of all these convex Minkowski sums. When the objects O_1 and O_2 are subdivided into m and n convex objects: $O_{1,i}$ $(1 \leq i \leq m)$ and $O_{2,j}$ $(1 \leq j \leq n)$, respectively, the resulting Minkowski sum $O_1 \oplus O_2$ is a union of mn convex Minkowski sums: $\cup_{1 \leq i \leq m, 1 \leq j \leq n} O_{1,i} \oplus O_{2,j}$, the representation of which requires at least $O(mn)$ edges/faces. Nevertheless, the number of edges/faces in the polygonal/polyhedral boundary $\partial(O_1 \oplus O_2)$ may end up to be only a small fraction of $O(mn)$. Moreover, the convex decomposition can be applied to polygonal/polyhedral objects only. When a non-convex curved object has a concave edge/face, it is

impossible to decompose the object into a finite union of convex objects. For handling curved objects as well as improving space efficiency, it is necessary to develop an algorithm which can compute the Minkowski sum boundary without using the convex decomposition of input objects.

Guibas et al. [19] suggested such an algorithm for polygonal objects in the plane. They investigated some important properties of the planar graph of convolution edges. The graph is composed of closed loop(s) which may self-intersect; that is, there is no dangling convolution edge in the graph. The planar convolution graph subdivides the plane into many disjoint regions. Each region has a winding number that is defined by the tracing of all closed loops along appropriate directions. The Minkowski sum boundary is defined as the set of convolution edges which are adjacent to a region with winding number zero. An important problem is how to define the directions of convolution edges so that each closed loop has a well-defined direction. However, this problem has not been seriously considered, yet. Moreover, the winding number classification seems not to work properly for the detection of the Minkowski sum boundary when an input object has small interior holes. In this paper, we consider two planar curved objects and construct their convolution graph as a union of closed loops. Each convolution edge has a well-defined direction and each convolution loop also has a well-defined orientation which is compatible with the directions of its component convolution edges.

Ghosh [16, 17, 18] presented in great detail the concepts, algorithms, and data structures that support the Minkowski sum and decomposition for polygonal/polyhedral objects. Assuming exact computation of line/line, plane/line, and plane/plane intersections, the Minkowski operations can be implemented based on the algorithms presented in Ghosh [16, 17, 18]. Rational arithmetic can be used to support such exact computation. However, when we use floating point arithmetic (which introduces numerical errors), it is non-trivial to implement Ghosh's algorithms robustly. Moreover, in the curved case, it is impossible to support exact computation of the curve/curve, curve/surface, and surface/surface intersections even if we use rational arithmetic. Therefore, there are still many challenging problems that must be resolved for the realization of the very important Minkowski operations of Ghosh [16, 17, 18] in practical situations. That is, we need to develop robust and efficient algorithms that can support the Minkowski operations on freeform curved objects, while using floating point computation. In this paper, we consider the Minkowski sum of planar curved objects. The Minkowski decomposition can be implemented in a similar way. However, the extension to solid objects (bounded by freeform surfaces) is far beyond the scope of this paper.

Bajaj and Kim [3, 4] discussed various important issues in the Minkowski sum computation for planar curved objects bounded by algebraic curves and also that for convex solid objects bounded by algebraic surfaces. The most important steps in the Minkowski sum computation include: a compatible subdivision of input curves/surfaces, the generation of convolution curves/surfaces, and the elimination of redundant parts. The convolution curves/surfaces are represented by simultaneous systems of polynomial equations. Some auxiliary variables are used in the simultaneous polynomial equations. Implicit curve/surface equations can be obtained by eliminating the auxiliary variables using resultant methods. However, the elimination process takes a considerable amount of symbolic computation time and the resulting curve/surface equations may have many redundant components. Therefore, the convolution curves/surfaces must be trimmed to represent only the portions of curve segments and surface patches which appear on the Minkowski sum boundary. This trimming procedure is non-trivial for implicit curves/surfaces especially when they have high degree, which is indeed the case for convolution curves/surfaces. Consequently, the convolution curves/surfaces must be approximated by parametric curve segments and surface patches for further construction of the Minkowski sum boundary. This paper presents several methods to approximate the convolution curves with planar polynomial/rational curve segments.

2.2 Exact Convolution Curve Computation

Ghosh [15] demonstrated that Equation (5) has a closed-form solution in some special cases. One such case is shown in the following example:

Example 2.1 Let $C_1(t)$ and $C_2(s)$ be two ellipses defined by

$$C_1(t) = (a\cos t, b\sin t), \quad \text{and} \quad C_2(s) = (p\cos s, q\sin s). \tag{6}$$

In this case, Equation (5) is represented as follows:

$$aq\cos s\sin t = bp\sin s\cos t. \tag{7}$$

We have two solutions for the reparametrization $s(t)$:

$$s_1(t) = \arctan(k\tan t), \quad \text{and} \quad s_2(t) = \pi + \arctan(k\tan t), \tag{8}$$

where $k = \frac{aq}{bp}$. There are two possible candidates for $C_2(s(t))$ that are reparametrized by $s_1(t)$ and $s_2(t)$, respectively:

$$C_2(s_1(t)) = \left(\frac{p\cos t}{\sqrt{k^2\sin^2 t + \cos^2 t}}, \frac{qk\sin t}{\sqrt{k^2\sin^2 t + \cos^2 t}}\right), \tag{9}$$

and

$$C_2(s_2(t)) = \left(\frac{-p\cos t}{\sqrt{k^2\sin^2 t + \cos^2 t}}, \frac{-qk\sin t}{\sqrt{k^2\sin^2 t + \cos^2 t}}\right). \tag{10}$$

Clearly, only the curve $C_2(s_1(t))$ satisfies the condition of Equation (4), namely:

$$\langle C_1'(t), C_2'(s_1(t))\rangle > 0. \tag{11}$$

Thus, the convolution curve $(C_1 * C_2)(t)$ is given as follows:

$$(C_1 * C_2)(t) = \left(a\cos t + \frac{p\cos t}{\sqrt{k^2\sin^2 t + \cos^2 t}}, b\sin t + \frac{qk\sin t}{\sqrt{k^2\sin^2 t + \cos^2 t}}\right). \tag{12}$$

□

Figure 7 shows the convolution curve of two ellipses that is computed by Equation (12). Ghosh [15] also considered other special cases such as ellipse–cubic and special quartic–cubic convolution curves. In these special cases, the exact convolution curves can be computed analytically; however, they are not rational, in general.

Bajaj and Kim [3, 4] showed that, when input curves/surfaces are given as implicit or parametric curves/surfaces, their convolution curves/surfaces can be represented exactly as implicit algebraic curves/surfaces. However, the resulting algebraic degrees are too high to be useful in practice [3]. Moreover, the convolution curve/surface trimming is non-trivial for implicit curves/surfaces.

Kaul and Farouki [25] took a similar approach to that of Bajaj and Kim [3]. They have more detailed and elaborate discussions on various important issues in the construction of the Minkowski sum boundary.

Kohler and Spreng [27] considered special cases in which convolution curves can be represented exactly, using radicals. They suggested some numerical techniques to speed up the compatible curve subdivision and convolution curve computation. The degree of $C_2(s)$ is restricted to be lower than or equal to five. Then, one can compute the exact closed-form solutions of $s(t)$ in Equation (5) by using radical expressions. However, the solutions are not rational, in general. Furthermore, one has to determine the correct solution from at most four possible candidates of $s(t)$.

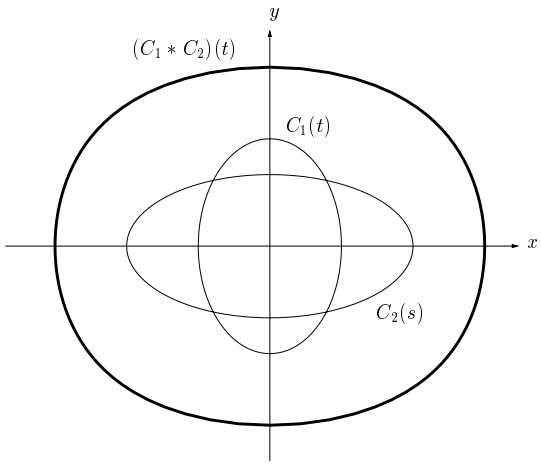


Figure 7: Convolution $(C_1 * C_2)(t)$ of two ellipses: $C_1(t) = (2\cos t, 3\sin t)$ and $C_2(s) = (4\cos s, 2\sin s)$, computed by Ghosh's method

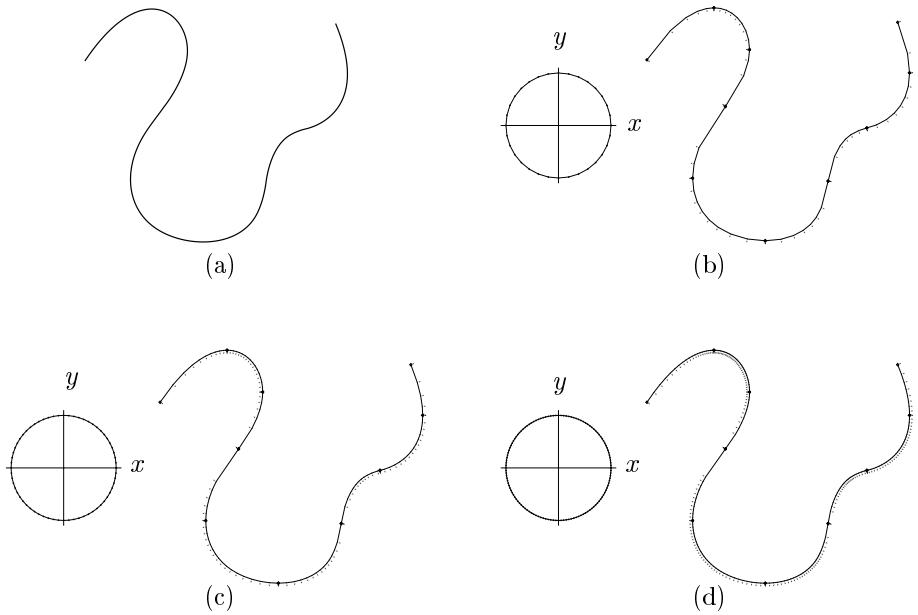


Figure 8: Gaussian approximations (GAP) of a cubic B-spline curve: (a) the original curve, (b) GAP corresponding to the 32 subdivision of the unit circle, (c) GAP to the 64 subdivision, and (d) GAP to the 128 subdivision.

2.3 Approximation Methods for Convolution Curve

Lee and Kim [29] suggested a method for approximating convolution curves with polygonal curves. In this method, each input curve is approximated by a sequence of discrete points (thus, forming a polygon) before the convolution computation. For a given set of evenly distributed normal directions, which is obtained by a regular subdivision of the unit circle, Lee and Kim [29] approximated each planar algebraic curve segment by a sequence of curve points. At each point, the curve gradient corresponds to one of the predefined normal directions (see Figure 8).

This curve approximation method is called *Gaussian Approximation* (GAP) and supports various primitive geometric operations such as offset, convolution, common tangent and distance computa-

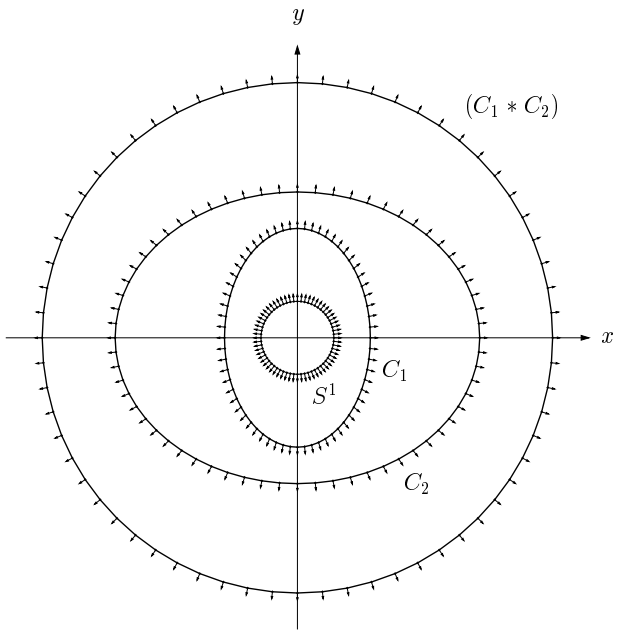


Figure 9: Gaussian approximations (GAP) of two ellipses C_1 and C_2 are generated based on a predefined circle subdivision S^1. $(C_1 * C_2)$ is the convolution curve of two ellipses.

tions [29]. An approximated convolution curve is computed as the sequence of vector sums generated by the pairs of curve approximation points corresponding to the same normal direction (see Figure 9). Since this approach is based on a linear approximation, a large number of discrete points is generated to approximate a convolution curve with high precision. In this paper, we propose several methods that approximate convolution curves with polynomial/rational spline curves. As a result, we can reduce output data significantly.

Ahn et al. [1] considered the more general problem of computing the boundary curve of a general sweep in the plane. In the general sweep, the sweeping object changes its shape and orientation dynamically while moving along a given trajectory. The convolution curve computation is a special case of general sweep boundary construction in which the sweeping object is restricted to a fixed shape and orientation. Ahn et al. [1] approximated the general sweep boundary curve with line segments. Ghosh [15] approximated the general sweep boundary with analytic curves under the assumption that the shape change of the moving object is negligible compared to its translational motion along the trajectory curve. The solution curves are not rational, in general. This paper presents several methods of approximating convolution curves with polynomial/rational curves. However, it is non-trivial to extend the methods of Ahn et al. [1] and Ghosh [15] so that one can approximate the general sweep boundary with polynomial/rational curves. Therefore, this problem remains an important open problem for future work.

Kaul and Farouki [25] suggested a piecewise linear approximation method for convolution curves. They generated a sequence of discrete points along the convolution curve on the fly, by computing Equation (5) for two compatible curve segments. That is, for a sampled sequence $\{t_i\}$ of the parameter t of $C_1(t)$, they computed the corresponding $\{s_i\}$ by solving the following equation:

$$y_2'(s_i)x_1'(t_i) - x_2'(s_i)y_1'(t_i) = 0. \tag{13}$$

We may interpolate the discrete values of s_i $(i = 1, \ldots, n)$ using a polynomial/rational function $s(t)$ so that $s(t_i) = s_i$, for each i. The resulting polynomial/rational curve $C_1(t)+C_2(s(t))$ is an approximation of the convolution curve $(C_1 * C_2)(t)$. This approximation method belongs to the reparametrization-

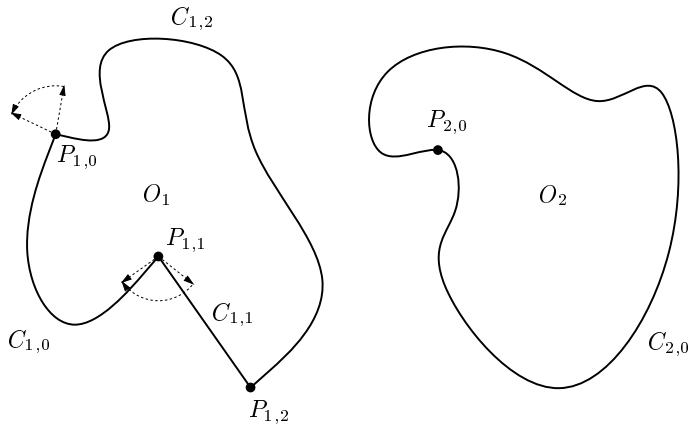


Figure 10: Examples of planar curved objects

based approach of convolution curve approximation methods to be discussed in Section 4.

3 Construction Algorithm for the Minkowski Sum

In this section, we review some basic concepts and present an algorithm for the boundary construction of the Minkowski sum of planar curved objects. Later sections will describe in more detail methods of computing/approximating convolution curves based on the materials presented in this section.

3.1 Planar Object, Boundary, and Normal

We start by defining the boundary of a planar curved object:

Definition 3.1 The boundary of a planar curved object O, denoted as ∂O, is represented by a connected sequence of piecewise smooth curve segments and their end points:

$$\partial O = \{P_0, C_0, P_1, C_1, ..., P_{n-1}, C_{n-1}\}, \tag{14}$$

where P_i and $P_{\mathrm{mod}(i+1,n)}$ are two end points of the curve segment C_i. We assume that the object may have cusps at vertices P_i only. Thus, none of the curve segment C_i has singularity except at the curve end points. □

Figure 10 shows examples of planar curved objects. Object O_1 consists of three boundary curve segments and O_2 has only one boundary curve segment.

In the convolution computation, each vertex P_i may be considered as an *edge* of the object, which has length zero, but has continuously changing normal vectors. Later, each curve segment is further subdivided into *convex* and *concave* curve segments by inserting all inflection points as extra vertices (see Section 3.2). The convexity of each edge (curve or vertex) is defined by as follows:

Definition 3.2 Let $C(t) = (x(t), y(t)), t_0 \le t \le t_1$, be a planar regular curve segment.

$$C(t) \text{ is an inflection point} \iff x'(t)y''(t) - x''(t)y'(t) = 0, \tag{15}$$

$$C(t), (t_0 \le t \le t_1), \text{ is convex} \iff x'(t)y''(t) - x''(t)y'(t) > 0, \text{ for } t_0 < t < t_1, \tag{16}$$

$$C(t), (t_0 \le t \le t_1), \text{ is concave} \iff x'(t)y''(t) - x''(t)y'(t) < 0, \text{ for } t_0 < t < t_1. \tag{17}$$

Let P_i be a vertex, and assume that $\epsilon > 0$ is an arbitrarily small number.

$$P_i \text{ is convex} \iff B_\epsilon(P_i) \cap O \text{ is smaller than a half of } B_\epsilon(P_i), \tag{18}$$

$$P_i \text{ is concave} \iff P_i \text{ is non-convex}, \tag{19}$$

where $B_\epsilon(P_i)$ is an ϵ-ball with center at P_i: $B_\epsilon(P_i) = \{P \mid \|P - P_i\| < \epsilon \}$. □

We assume that each curve segment $C(t)$, for $t_0 \le t \le t_1$, has no inflection point in the curve interior, i.e., $x'(t)y''(t) - x''(t)y'(t) \neq 0$, for $t_0 < t < t_1$. Then the curve segment $C(t)$ is either convex or concave. We need the following definition to introduce the *Gauss map* of an edge:

Definition 3.3 For a curve segment $C(t) = (x(t), y(t))$, $t_0 \le t \le t_1$, the *unit normal vector field* of $C(t)$ is defined by

$$N(t) = \frac{(y'(t), -x'(t))}{\sqrt{x'(t)^2 + y'(t)^2}} \in S^1, \quad t_0 \le t \le t_1, \tag{20}$$

where S^1 is the unit circle. □

We assume that the boundary curve segments of a planar curved object are oriented in counterclockwise order. That is, the normal vectors of a planar curve are pointing to the right-hand side of the curve advancing direction.

Definition 3.4 The *Gauss map* $\mathcal{N}(C)$ of a curve segment $C(t) = (x(t), y(t))$, $t_0 \le t \le t_1$, is a unit circular arc defined by:

$$\mathcal{N}(C) = \{N(t) \mid t_0 \le t \le t_1 \} \subset S^1, \tag{21}$$

where S^1 is the unit circle. For a vertex P_i, let $N_{i-1}(t_{i-1,1})$ and $N_i(t_{i,0})$ be the unit normal vectors of $C_{i-1}(t)$, $t_{i-1,0} \le t \le t_{i-1,1}$, and $C_i(t)$, $t_{i,0} \le t \le t_{i,1}$, respectively, at the common vertex P_i $(= C_{i-1}(t_{i-1,1}) = C_i(t_{i,0}))$. The *Gauss map* $\mathcal{N}(P_i)$ of a convex (resp. concave) vertex P_i is defined as the unit circular arc which connects $N_{i-1}(t_{i-1,1})$ and $N_i(t_{i,0})$ in counterclockwise (resp. clockwise) direction on the unit circle S^1. □

A different choice of orientation (i.e., counterclockwise/clockwise direction) for the Gauss maps of convex/concave vertices implies that the Gauss maps generate unit circular arcs of length less than or equal to π. In Figure 10, $P_{1,0}$ and $P_{1,1}$ are convex and concave vertices, respectively. The dashed circular arcs represent the Gauss maps of the vertex edges. Note that the arc direction of $\mathcal{N}(P_{1,0})$ (resp. $\mathcal{N}(P_{1,1})$) is counterclockwise (resp. clockwise). Finally, we define the *compatible pair* of edges:

Definition 3.5 Let e_1 and e_2 be two edges on ∂O_1 and ∂O_2, respectively. Two edges e_1 and e_2 are *compatible* if and only if $\mathcal{N}(e_1) = \mathcal{N}(e_2)$. □

The convexity of an edge is not important in the definition of compatibility. Note that an edge is either a curve segment or a vertex.

3.2 Compatible Subdivision

For the sake of simplicity, we first assume that two input objects, O_1 and O_2, are bounded by smooth curves, $C_1(t)$ and $C_2(s)$, respectively. That is, $C_1(t) = (x_1(t), y_1(t))$, $t_0 \le t \le t_1$, and $C_2(s) = (x_2(s), y_2(s))$, $s_0 \le s \le s_1$, are closed regular curves, and they have G^1-continuity at $C_1(t_0) = C_1(t_1)$ and $C_2(s_0) = C_2(s_1)$, respectively. Then the objects O_1 and O_2 have no cusp on their boundaries. In this subsection, we consider how to subdivide the two boundary curves $C_1(t)$ and $C_2(s)$ into compatible subsegments by *hodograph subdivision* as shown in Figure 11.

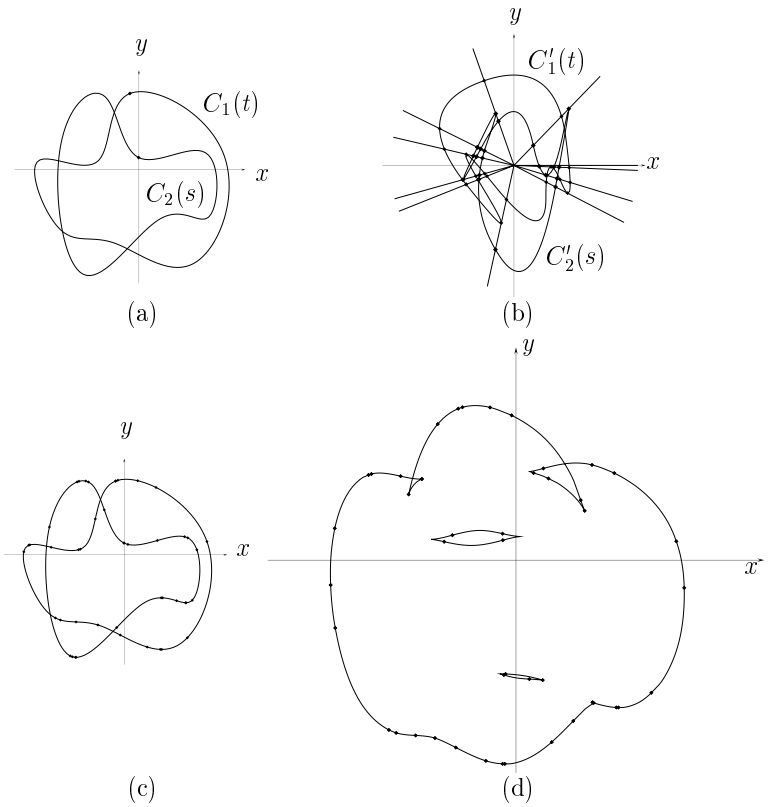


Figure 11: Hodograph subdivision

Let $C_1(\bar{t}_i)$, $(0 \leq i < \bar{m})$, and $C_2(\bar{s}_j)$, $(0 \leq j < \bar{n})$, be all the inflection points of $C_1(t)$ and $C_2(s)$, respectively. The inflection points can be computed based on Definition 3.2 using symbolic and numeric computation tools [8]. By inserting each inflection point as a new vertex, the curve is subdivided into convex and concave subsegments. For each inflection point, $C_1(\bar{t}_i)$ (resp. $C_2(\bar{s}_j)$), the ray $\overline{OC_1'(\bar{t}_i)}$ (resp. $\overline{OC_2'(\bar{s}_j)}$) emanating from the origin and passing through the point $C_1'(\bar{t}_i)$ (resp. $C_2'(\bar{s}_j)$) is tangent to the hodograph $C_1'(t)$ (resp. $C_2'(s)$) (see Equation 15). The set of rays:

$$\left\{\overline{OC_1'(\bar{t}_i)}, \overline{OC_2'(\bar{s}_j)}, \overline{OC_1'(t_0)}, \overline{OC_2'(s_0)} \mid 0 \leq i < \bar{m},\ 0 \leq j < \bar{n}\right\} \tag{22}$$

constitutes a set of sector-form regions R_k, $(0 \leq k < l)$, which is a partition of the 2D plane R^2 (see Figure 11(b)).

After computing all the inflection points, the hodographs $C_1'(t)$ and $C_2'(s)$ are subdivided into piecewise curves $C_{1,i}'(t)$, $(0 \leq i < m)$, and $C_{2,j}'(s)$, $(0 \leq j < n)$, at the intersection points with the rays in the set of Equation (22). The intersection points of a line and a curve can also be computed using symbolic and numeric computation tools. The original curves, $C_1(t)$ and $C_2(s)$, are subdivided at the parameter values corresponding to the end points of the hodograph curve segments $C_{1,i}'(t)$ and $C_{2,j}'(s)$, respectively (see Figure 11(c)).

Each curve segment, $C_{1,i}(t)$ or $C_{2,j}(s)$, is recorded in the sector-form region R_k which includes the corresponding hodograph curve $C_{1,i}'(t)$ or $C_{2,j}'(s)$. All the pairs $(C_{1,i}(t), C_{2,j}(s))$ that are recorded in the same sector-form region are compatible pairs; that is, $\mathcal{N}(C_{1,i}) = \mathcal{N}(C_{2,j})$. Figure 11(d) shows the resulting convolution curves generated from $C_1(t)$ and $C_2(s)$.

In convolution curve computation, it is convenient to assume that each curve $C(t)$ is convex/concave and its Gauss map $\mathcal{N}(C)$ has length less than π. For each normal vector $N \in \mathcal{N}(C)$, we can then

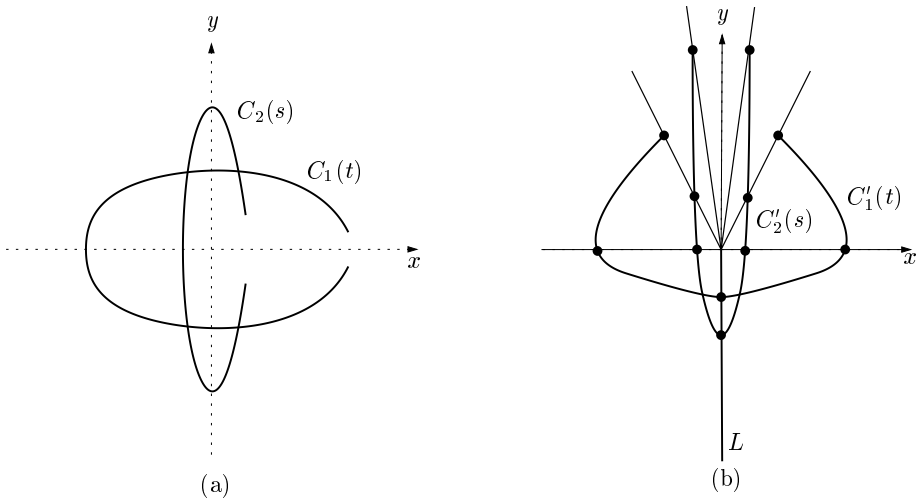


Figure 12: (a) Two planar curves and (b) their hodograph subdivisions (also with additional subdivisions along the x- and y-axes).

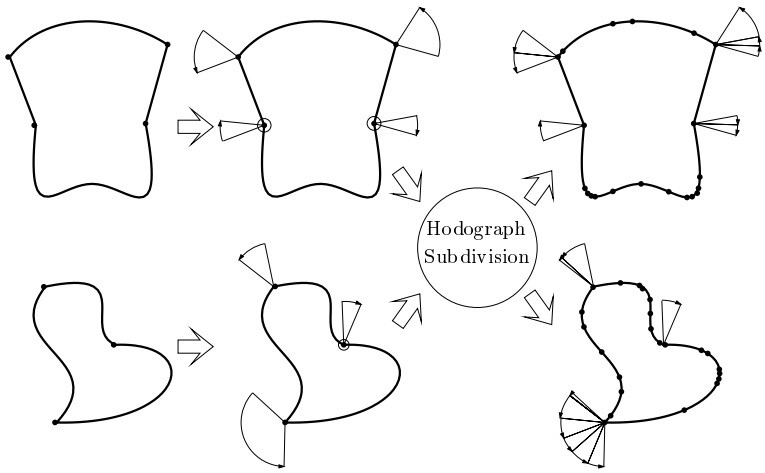


Figure 13: Hodograph subdivision of curves with cusps

compute a unique curve point $C(t)$ that corresponds to the normal vector N as follows:

$$\langle C'(t), N \rangle = 0. \tag{23}$$

Such subdivision of $C_1(t)$ and $C_2(s)$ can be done by inserting extra vertices at all inflection points and at the intersection points of their hodographs with some extra rays. One simple method is to use the x- and y-axes as extra rays (see Figure 12). Figure 12 shows two planar curves and their hodograph subdivisions so that each Gauss map belongs to a quadrant of the unit circle S^1.

In general situations, the boundary curves may have cusps at vertices; that is, the object boundary may not be G^1-continuous (see Figure 13). Then the algorithm becomes more complex. Each cusp point may be treated as an edge with a sector-form Gauss map. Each concave cusp point (represented as $\odot$ in Figure 13) is treated in a similar way. In the hodograph subdivision process, the Gauss map of each point edge is also subdivided by the rays from the origin as shown in Figure 11(b).

3.3 Computing Convolution Edges

After compatible subdivision, a convolution edge is computed for each pair of compatible edges. As mentioned in the previous subsection, the compatible pair may be a curve–curve or curve–point pair. For simplicity of explanation, we consider the case of polygonal objects first.

3.3.1 Polygonal Objects

Given two polygonal objects, we apply the following three rules [19, 33] to compute the convolution edges:

1. For a pair of points, P_1 and P_2, such that $\mathcal{N}(P_1) \cap \mathcal{N}(P_2) \neq \emptyset$,

$$P_1 * P_2 = P_1 + P_2.$$

2. For a point P_1 and a line segment $\overline{P_2P_3}$ such that $\mathcal{N}(\overline{P_2P_3}) \subset \mathcal{N}(P_1)$,

$$P_1 * \overline{P_2P_3} = \overline{(P_1 + P_2)(P_1 + P_3)}.$$

3. For a pair of compatible line segments $\overline{P_1P_2}$ and $\overline{P_3P_4}$ such that $\mathcal{N}(\overline{P_1P_2}) = \mathcal{N}(\overline{P_3P_4})$,

$$\overline{P_1P_2} * \overline{P_3P_4} = \overline{(P_1 + P_3)(P_2 + P_4)}.$$

The above rules imply that the strict compatible subdivision (described in Section 3.2) is not necessary for the convolution of two polygonal objects. Figure 14 shows an example of convolution edge computation for two convex polygonal objects. Each convolution edge C_i is computed based on the above three rules. Clearly, we can ignore Rule 1 because it generates convolution vertices only.

For non-convex polygonal objects, there are some subtle cases which need careful treatment. In Figure 15, two vertices a_7 and b_7 are concave vertices. The Gauss map of each concave vertex has a clockwise orientation. For example, the normal angle of a_7 changes from 280° to 260°, instead of changing from 260° to 280°. The clockwise orientation has an important implication in the determination of the orientation for each convolution edge generated from a compatible pair of a concave vertex and a line segment.

In Figure 15, each convolution edge has its edge direction inherited from that of component edge(s). For example, the convolution edge $c_5 = a_4 * b_7$ is obtained from the parallel translation of a_4 by a vector b_7. The edge direction of c_5 in this figure is the same as that of a_4. Similarly, the convolution edges $c_{12} = a_8 * b_7$ and $c_{15} = a_7 * b_6$ are given the same directions as those of a_8 and b_6, respectively. Unfortunately, this strategy does not produce a consistent orientation for each closed loop of convolution edges. For example, the edge direction of c_5 is inconsistent with those of c_4 and c_6. Moreover, c_{12} and c_{15} have inconsistent directions with those of c_{13} and c_{14}.

To make the orientation of a convolution loop consistent with the directions of all its component convolution edges, we take the simple strategy of reversing the direction of each convolution edge generated from a concave vertex and a line segment. (We have a similar problem in computing the convolution edge for each pair of compatible curve segments; more details will be discussed in Section 3.3.2.) Figure 16(a) shows two convolution loops which have consistent orientations. The loops subdivide the plane R^2 into four disjoint connected regions, each of which is assigned a unique winding number. The winding number provides an important theoretical tool in characterizing the Minkowski sum boundary from the superset consisting of all convolution edges; namely, a convolution edge belongs to the Minkowski sum boundary if and only if it is on the boundary of a region with winding number zero. (This characterization holds under the assumption that two input objects have no holes.)

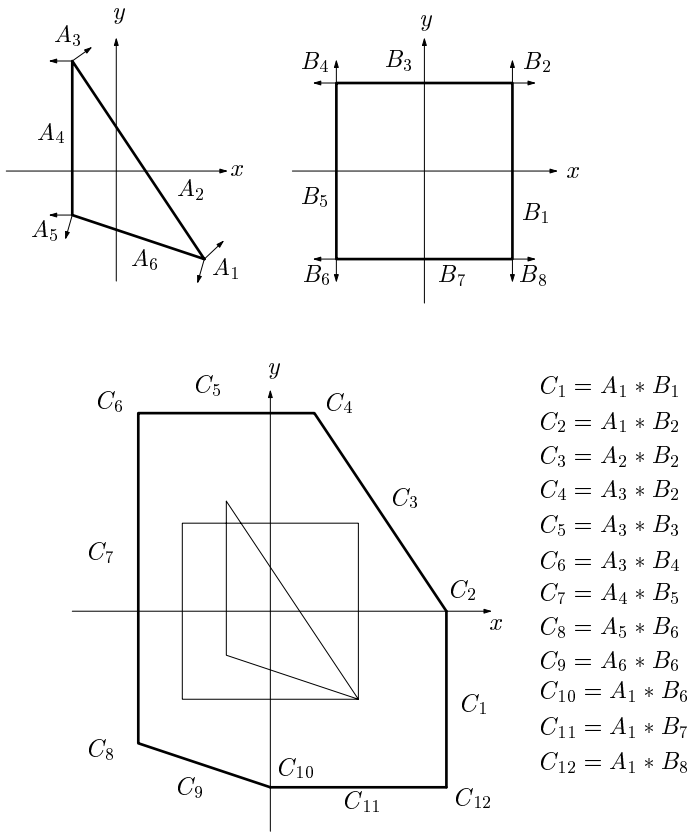


Figure 14: Convolution computation for convex polygons

Guibas et al. [19] introduced the concept of winding number to computational geometry. A geometric interpretation of winding number in the convolution curve arrangement may be given as follows. Consider two simply-connected planar objects O_1 and O_2 bounded by $C_1(t)$ and $C_2(s)$, respectively, and the convolution curve $(C_1 * C_2)(t)$ which forms a superset of the Minkowski sum boundary $\partial(O_1 \oplus O_2)$. When the reversed object $-O_2$ is translated so that its center is located in a region of winding number k (for the convolution $C_1 * C_2$), the two objects O_1 and $-O_2$ intersect in k disjoint regions. Figure 16(b) shows four different translated instances of the reversed object $-O_2$, where the reference point of each instance is located in a region of winding number k $(= 0, 1, 2)$. Note that each instance of $-O_2$ intersects with O_1 in k disjoint regions (shown in gray). When we move $-O_2$ (with a fixed orientation) while its reference point is contained in the same region of winding number k, the number of disjoint regions in $O_1 \cap (-O_2)$ is always fixed to k. The *C-space* obstacle boundary, $\partial(O_1 \oplus O_2)$, is in between two different configurations: one for collision and the other for collision-free. Therefore, it is clear that only the convolution edges adjacent to regions of winding number zero can contribute to the Minkowski sum boundary $\partial(O_1 \oplus O_2)$. Moreover, each point on the Minkowski sum boundary must be adjacent to a region of winding number zero.

When the object O_2 is a circle with its center at the origin, we have $-O_2 = O_2$ and the winding number theory can be applied to the boundary construction of the offset of O_1, that is, to the construction of $\partial(O_1 \oplus O_2)$. It is interesting to note that the offset boundary classification of Hansen and Arbab [21] is equivalent to the winding number theory which has been known in computational geometry for many years [19]. In the above discussion of the winding number theory, we have restricted the input objects to those with no holes. When an object's holes are sufficiently large for the other object to be totally contained in each hole, we can apply the same characterization to classify the convolution edges which appear on the Minkowski sum boundary. In NC machining, small holes are

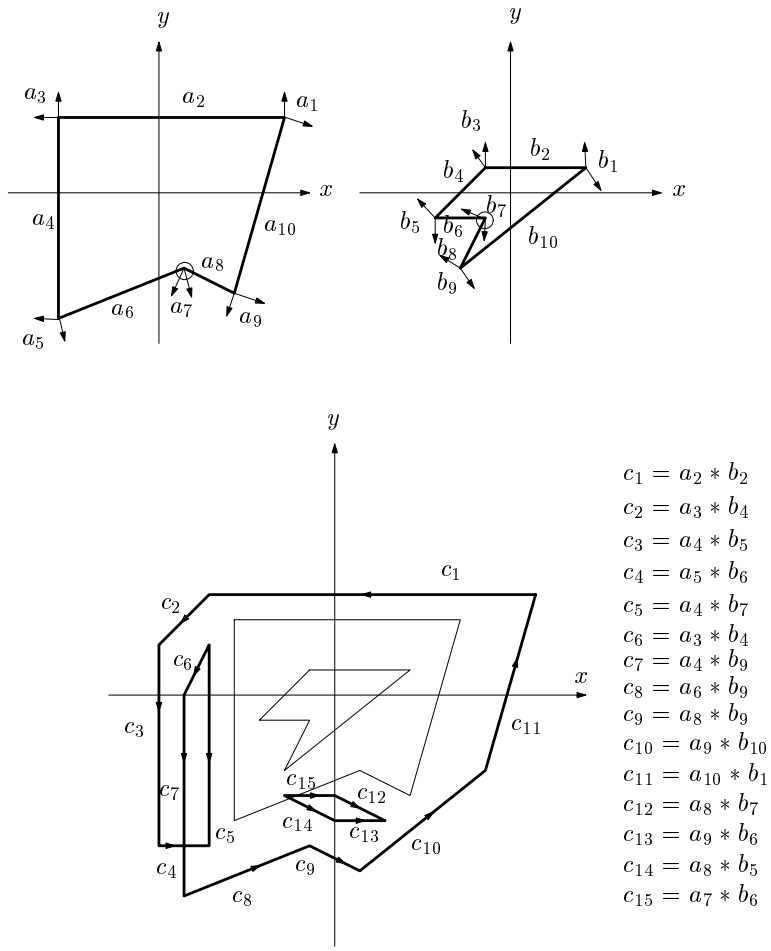


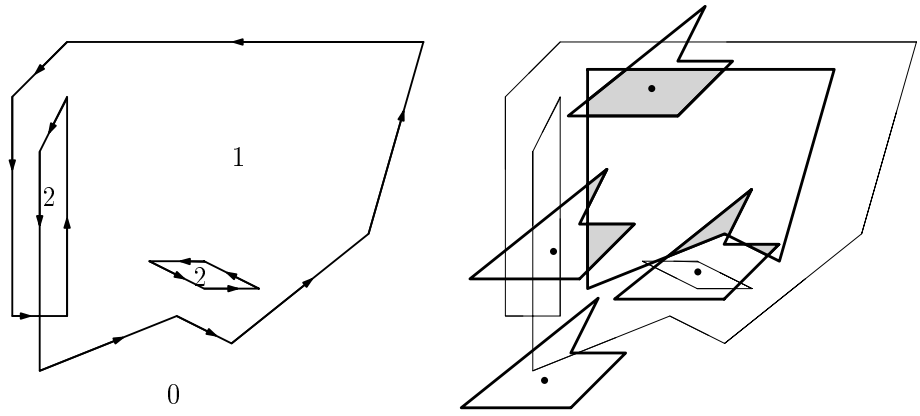
Figure 15: Convolution computation for non-convex polygons: each convolution edge direction is inherited from its component edge directions.

not allowed since they introduce gouging when they do not contain the machining tool completely. Therefore, the winding number theory can be applied with no restriction to the offset operation in NC pocket machining.

The winding number technique provides a theoretical solution to the construction of the Minkowski sum boundary. However, it is still quite doubtful whether this technique can contribute to the robustness and efficiency of a construction algorithm. When we deal with non-polygonal convolution curves, there has been no known implemented algorithm that can robustly determine the arrangement of convolution curves. The determination of curve arrangement is the most crucial step in computing the winding number of each connected region.

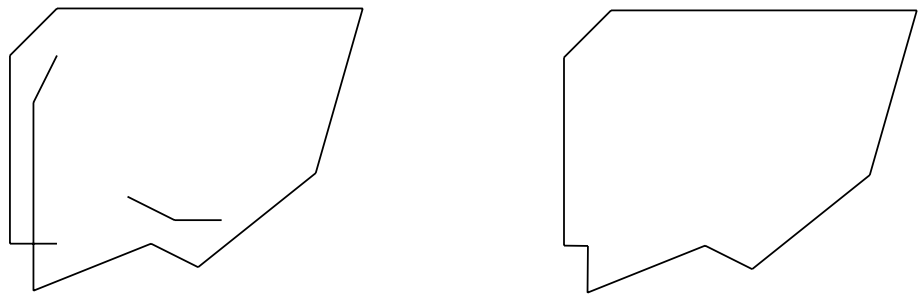
In dealing with polygonal convolution edges (or polygonal approximations of convolution curves), a simple way to implement a robust arrangement of line segments is to use exact rational arithmetic. However, this strategy does not provide an efficient solution to the arrangement of line segments, especially when a large number of line segments are used to approximate curved convolution edges. Guibas et al. [20] presented an efficient technique that uses floating-point arithmetic and determines the arrangement of line segments robustly. In this paper, we use a similar technique that was also implemented in Ahn et al. [1].

The convolution edges generated from at least one concave vertex do not appear on the final boundary of a Minkowski sum. Moreover, they have no contribution to the robustness and efficiency of an arrangement algorithm for convolution edges. Consequently, we can simply remove all convolution



(a) (b)

Figure 16: Consistent edge directions and the winding number of each region



(a) (b)

Figure 17: Elimination of redundant parts in the convolution of concave polygons

edges generated from at least one concave vertex (Figure 17(a)), and compute the Minkowski sum boundary by eliminating other redundancies (Figure 17(b)). In fact, the elimination of as many redundant convolution edges as possible in a preprocessing step improves the robustness and efficiency of the construction algorithm for a Minkowski sum boundary. In Section 6, we will discuss some preprocessing techniques which compute simpler Minkowski sums and eliminate many redundant convolution edges based on them.

3.3.2 Planar Curved Objects

The convolution rules for planar curved objects are similar to those for polygonal objects:

1. For a point P and a curve segment C such that $\mathcal{N}(P) = \mathcal{N}(C)$ (i.e., they are compatible),

$$P * C = P + C.$$

2. For a pair of compatible curve segments C_1 and C_2, i.e., $\mathcal{N}(C_1) = \mathcal{N}(C_2)$,

$$(C_1 * C_2)(t) = C_1(t) + C_2(s(t)),$$

where

$$C_1'(t) \parallel C_2'(s(t)) \quad \text{and} \quad \langle C_1'(t), C_2'(s(t)) \rangle > 0.$$

In the curved case, the convexity of each curve segment is important. When the convexities of two compatible edges are different, the determination of convolution edge direction becomes complex since it depends on the relative curvature distribution of each component curve. Assume that $C_1(t)$ and $C_2(s)$ are two compatible curve segments which are concave and convex, respectively (see Figures 18(a)–(b)). Moreover, assume that $C_1(t)$ and $C_2(s)$ are arc-length parametrized by t and s, respectively. The first derivative $(C_1 * C_2)'(t)$ is computed as follows:

$$(C_1 * C_2)'(t) = C_1'(t) + C_2'(s(t))s'(t).$$

Since the convexities of $C_1(t)$ and $C_2(s)$ are different, the reparametrization $s(t)$ decreases as the parameter t increases; that is, we have $s'(t) < 0$, for all t. When the curvature of $C_1(t)$ is larger than that of $C_2(s(t))$, the speed of $s(t)$ is larger than 1 and we have $s'(t) < -1$ (see Figures 18(d)–(e)). Therefore, the direction of $(C_1 * C_2)'(t)$ is opposite to that of $C_1'(t)$, and it is parallel to that of $s'(t)C_2'(s(t))$ (equivalently, to that of $-C_2'(s(t))$). This explains why we need to reverse the edge direction in each convolution edge that is generated from a concave vertex and a line segment (see Section 3.3.1).

A convolution curve $(C_2 * C_1)(s)$ having the same curve trace as that of $(C_1 * C_2)(t)$ can be constructed by switching the roles of $C_1(t)$ and $C_2(s)$:

$$(C_2 * C_1)(s) = C_1(t(s)) + C_2(s),$$

where

$$C_1'(t(s)) \parallel C_2'(s) \quad \text{and} \quad \langle C_1'(t(s)), C_2'(s) \rangle > 0.$$

The first derivative $(C_2 * C_1)'(s)$ is then computed as follows:

$$(C_2 * C_1)'(s) = C_1'(t(s))t'(s) + C_2'(s).$$

When the curvature of $C_1(t(s))$ is larger than that of $C_2(s)$, the speed of $t(s)$ is smaller than 1 and we have $-1 < t'(s) < 0$ (see Figures 18(f)–(g)). Therefore, the direction of $(C_2 * C_1)'(s)$ is the same as that of $C_2'(s)$. Note that this direction is opposite to that of $(C_1 * C_2)'(t)$. This may look self-contradictory. However, note that the two convolution curves $(C_1 * C_2)(t)$ and $(C_2 * C_1)(s)$ have the same curve trace; nevertheless, they are parametrized in opposite directions when the convexities of $C_1(t)$ and $C_2(s)$ are different (see Figures 18(d) and 18(f)). Therefore, we have to select the convolution edge direction from the two opposite directions of $(C_1 * C_2)(t)$ and $(C_2 * C_1)(s)$. In the convolution graph of Figure 18(c), we can easily notice that the edge direction of $(C_1 * C_2)(t)$ will produce a consistent orientation for the convolution loop.

When we slightly bend a line segment into a concave circular arc with a very large radius and also slightly round a concave vertex into a concave circular arc with a very small radius, the resulting convolution edge (i.e., a circular arc with a large radius) must have almost the same curve shape and edge direction as the convolution linear edge generated by the line segment and the concave vertex. That is, the convolution edge must have the opposite direction to that of the two input concave edges. Figures 19(a)–(b) show two input objects. The boundary of each object consists of two line segments and a concave circular arc. The convolution edge of two concave circular arcs is also a circular arc, the radius of which is given by the addition of the radii of two input circular arcs. Note that the convolution edge direction is opposite to that of the two input concave edges (see Figure 19(c)).

When we examine the winding number of each connected region in the planar convolution graph, we find that the winding number of the region to the left of each convolution edge is one larger than that of the region to the right (see Figure 19(c)). In particular, the convolution edge generated by two compatible concave edges reverses its edge direction from that of the two input edges so that it correctly reflects the more complex interference between two input objects in its left rather than

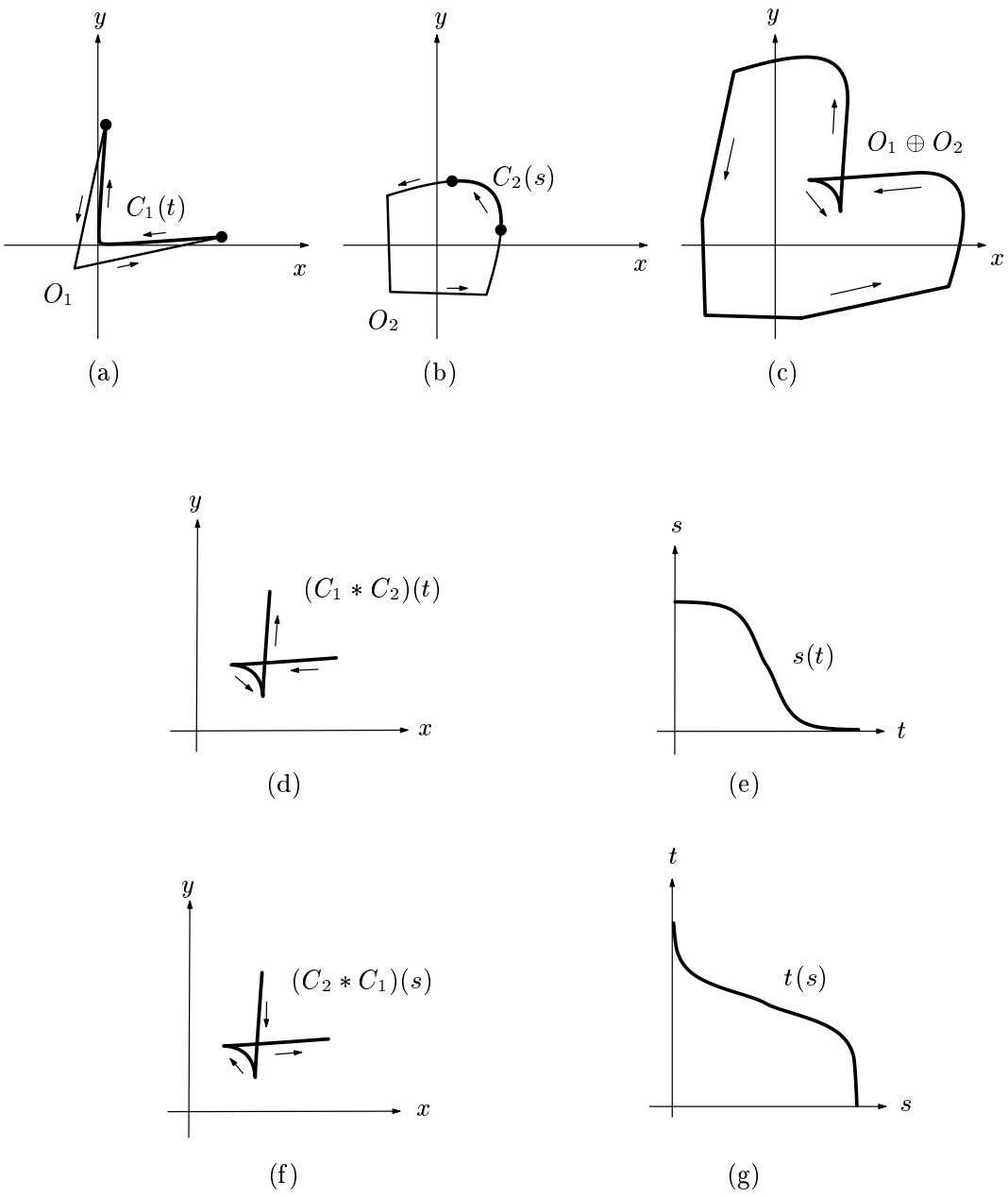


Figure 18: Convolution curve for a compatible pair of convex-concave edges

its right-hand side. Note that the right-hand side of such a convolution edge also belongs to the Minkowski sum interior (see Figure 19(c)). Consequently, the convolution edges generated by the compatible pairs of concave edges cannot appear in the Minkowski sum boundary. Using a similar argument, we can also show that the convolution edges generated by at least one concave vertex do not contribute to the Minkowski sum boundary. Figures 19(d)–(f) show the relationship between the winding number of a region in the convolution graph and the number of disjoint regions in $O_1 \cap (-O_2)$.

In computing the Minkowski sum of two curved objects, we cannot simply ignore the convolution curve segments generated from convex–concave (or concave–convex) edge pairs. They may also contribute to the final boundary of a Minkowski sum [3]. However, a convolution curve segment is redundant when it is generated by a convex curve $C_1(t)$ with smaller curvature than its compatible concave curve $C_2(s)$ (see Section 6 for more details).

Figure 20(a) shows two non-convex planar curved objects. Their untrimmed convolution curves are shown in Figure 20(b) and the Minkowski sum boundary of the two curved objects is shown

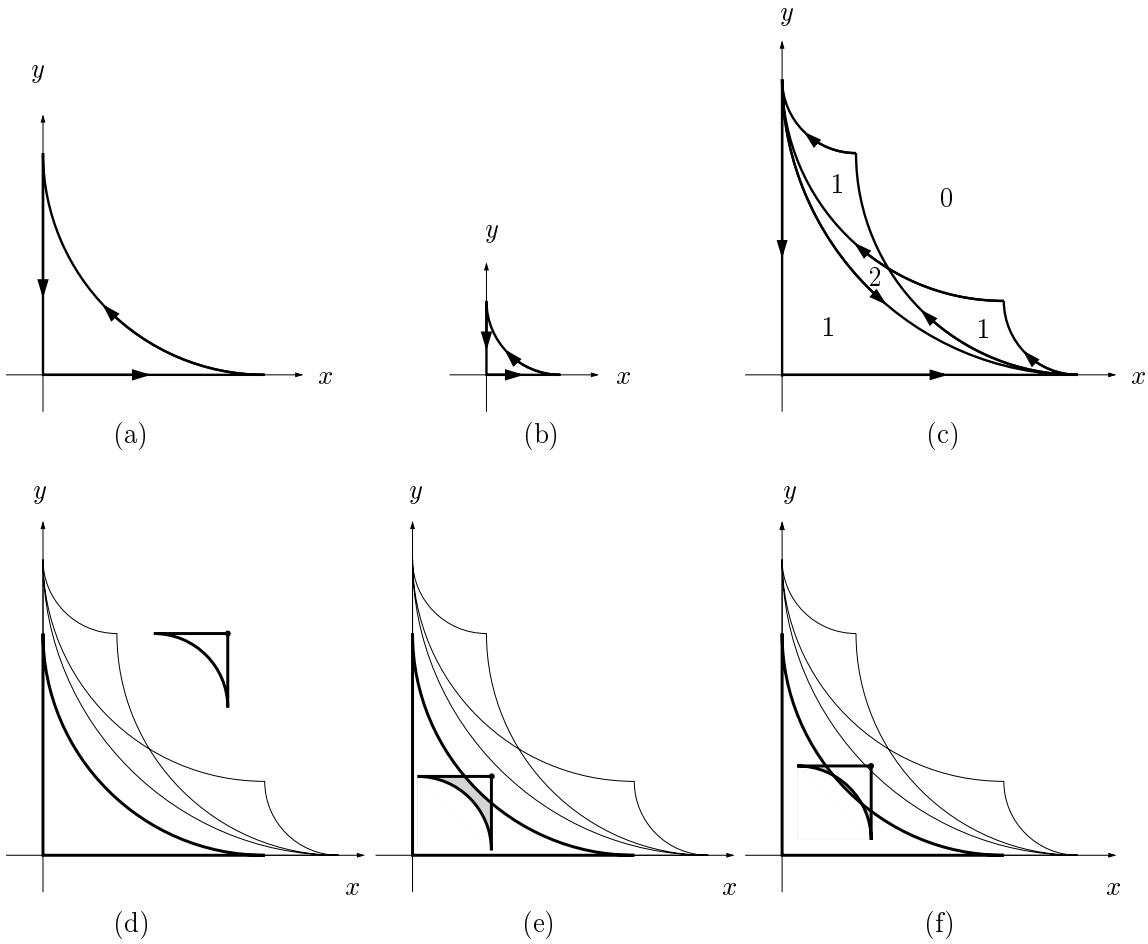


Figure 19: Convolution curve for a compatible pair of concave-concave edges

in Figure 20(c). Figure 20(b) and the table in Figure 20 represent different types of convolution curves. Comparing the curve/vertex types of two compatible input edges and their convexities, the convolution edge can be classified as in the table of Figure 20. For example, Type 3 convolution curves are generated from the pairs of concave–concave curve segments and the pairs including at least one concave vertex. Convolution curves of Type 3 cannot appear on the Minkowski sum boundary. However, we must consider all other types of convolution curve segments (see Figure 20(b)). An algorithm for eliminating redundant convolution curve segments will be described in Section 6.

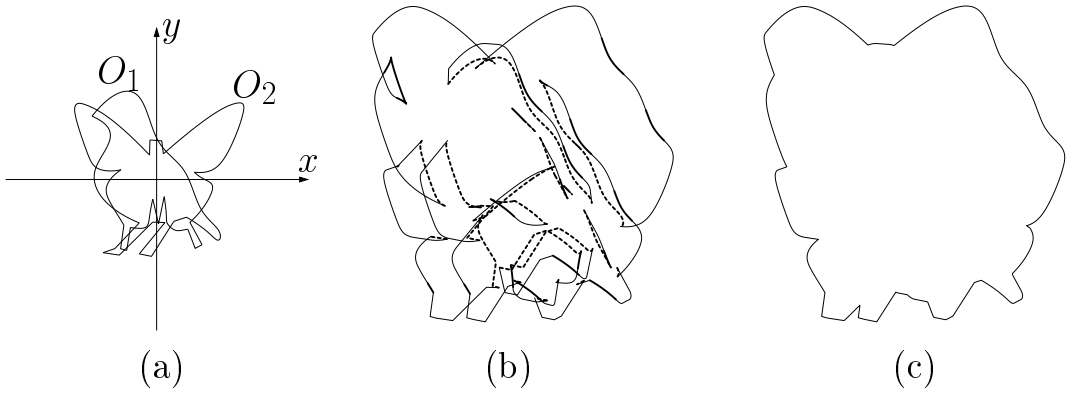
4 Convolution Curve Approximation Methods

In this section, we present several methods to compute a convolution curve segment for each pair of compatible curve segments. These methods can be classified into four types of approach: *control point based*, *interpolation based*, *quadratic curve approximation based*, and *reparametrization based*. All these convolution curve approximation methods are conceptually similar to offset curve approximation methods [10].

Let $C_1(t)$, $t_0 \le t \le t_1$, and $C_2(s)$, $s_0 \le s \le s_1$, denote two compatible freeform curve segments. Without loss of generality, we may assume:

$$C_1'(t_0) \parallel C_2'(s_0) \quad \text{and} \quad C_1'(t_1) \parallel C_2'(s_1). \tag{24}$$

Moreover, let $(C_1 *^a C_2)(t)$ denote an approximation curve of $(C_1 * C_2)(t)$.



edge of O_1 \ edge of O_2		curve segment		vertex	
		convex	concave	convex	concave
curve segment	convex	Type 1	Type 2	Type 1	Type 3
	concave	Type 2	Type 3	Type 2	Type 3
vertex	convex	Type 1	Type 2	-	-
	concave	Type 3	Type 3	-	-

Figure 20: Computing convolution edges for curved objects: (a) two planar objects O_1 and O_2, (b) Type 1 (light solid curves), Type 2 (bold solid curves), and Type 3 (bold dashed curves) convolution curves (see the table), and (c) the Minkowski sum boundary

4.1 Control Point Based Method (CTC)

The control point based method is the simplest method. This method does not consider the relationship between the normal directions of two input curves. Therefore, this method does not generate

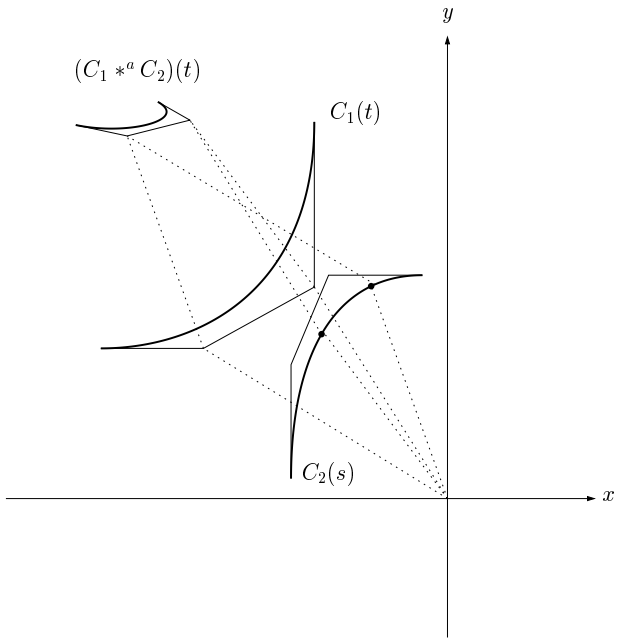


Figure 21: CTC approximation $(C_1 *^a C_2)(t)$ of $C_1(t)$ and $C_2(s)$. Two internal control points P_1 and P_2 are translated by $C_2(\hat{\xi}_1)$ and $C_2(\hat{\xi}_2)$ (two dots on $C_2(s)$).

very precise approximations compared with other approximation methods. However, the control point based method uses simple arithmetic operations only. The implementation is also quite straightforward.

Let $C(t)$ be a B-spline curve of degree d with n control points $\{P_i\}$, $0 \leq i < n$, and knot vector $\{k_i\}$, $0 \leq i < d+n+1$. The following sequence $\{\xi_i\}$, $0 \leq i < n$, represents *node*, or Greville abscissae [12], parameter values of $C(t)$:

$$\xi_i = \frac{\sum_{j=i+1}^{i+d} k_j}{d}. \tag{25}$$

Hence, a node parameter value is an average of d consecutive knots in $\{k_i\}$. Each control point P_i of $C(t)$ is associated with the node ξ_i. $C(\xi_i)$ is typically close to P_i; however, it is not the closest point of $C(t)$ to P_i, in general.

Let P_i and ξ_i, $0 \leq i < n$, be the control points and the node parameter values of $C_1(t)$. An approximated convolution curve can be computed by translating each control point P_i by $C_2(\hat{\xi}_i)$, where

$$C_1'(\xi_i) \parallel C_2'(\hat{\xi}_i), \quad s_0 \leq \hat{\xi}_i \leq s_1. \tag{26}$$

The unique parameter $\hat{\xi}_i$ can be computed by solving the following equation (see Section 3.2):

$$\left\langle N_1(\xi_i), C_2'(\hat{\xi}_i) \right\rangle = 0, \tag{27}$$

where $N_1(t)$ is the unit normal vector field of $C_1(t)$. This method of convolution curve approximation is called CTC (Control point Translation Convolution). This approximation method may be considered as a generalization of Cobb's offset approximation method [5]. Figure 21 shows an example of CTC approximation.

4.2 Interpolation Based Methods

A natural approach to the approximation of a convolution curve is to interpolate the convolution points which are computed from some sample points on the input curves. Although interpolation

based methods need intensive computation, they generate better approximations than control point based methods.

4.2.1 Convolution Using Least Squares Approximation (LSC, BIC)

LSC (Least Squares Convolution) and BIC (B-spline Interpolation Convolution) methods compute the convolution points at finite sample parameters. After that, they approximate or interpolate these discrete points with spline curves. Let $\chi_0, \chi_1, ..., \chi_{m-1}$, be m finite sample parameters of $C_1(t)$, where $\chi_0 = t_0$ and $\chi_{m-1} = t_1$. For each χ_i, the exact convolution point $(C_1 * C_2)(\chi_i)$ is computed as follows:

$$(C_1 * C_2)(\chi_i) = C_1(\chi_i) + C_2(\hat{\chi}_i), \quad s_0 \leq \hat{\chi}_i \leq s_1, \tag{28}$$

where

$$C_1'(\chi_i) \parallel C_2'(\hat{\chi}_i). \tag{29}$$

The parameters $\hat{\chi}_i$ can be computed using Equation (27).

There are many well-known methods for the approximation or interpolation of a given point set by a B-spline curve [23]. A set of convolution points, $\{(C_1 * C_2)(\chi_i) \mid i = 0, 1, 2, ..., m-1\}$, can be either (i) approximated by a B-spline curve using the least squares method (LSC) or (ii) interpolated by a B-spline curve (BIC).

4.2.2 Convolution Using Hermite Interpolation (HIC)

The derivative of an exact convolution curve is computed as follows:

$$\frac{\partial (C_1 * C_2)}{\partial t}(t) = \frac{\partial C_1}{\partial t}(t) + \frac{\partial C_2}{\partial s}(s(t))\frac{\partial s}{\partial t}(t). \tag{30}$$

¿From Equation (3), we have

$$\frac{\partial x_1}{\partial t}\frac{\partial y_2}{\partial s} - \frac{\partial x_2}{\partial s}\frac{\partial y_1}{\partial t} = 0. \tag{31}$$

By differentiating Equation (31) with respect to t, we have

$$\frac{\partial^2 x_1}{\partial t^2}\frac{\partial y_2}{\partial s} + \frac{\partial x_1}{\partial t}\frac{\partial^2 y_2}{\partial s^2}\frac{\partial s}{\partial t} - \frac{\partial^2 x_2}{\partial s^2}\frac{\partial s}{\partial t}\frac{\partial y_1}{\partial t} - \frac{\partial x_2}{\partial s}\frac{\partial^2 y_1}{\partial t^2} = 0, \tag{32}$$

and

$$\frac{\partial s}{\partial t} = \frac{\frac{\partial x_2}{\partial s}\frac{\partial^2 y_1}{\partial t^2} - \frac{\partial^2 x_1}{\partial t^2}\frac{\partial y_2}{\partial s}}{\frac{\partial x_1}{\partial t}\frac{\partial^2 y_2}{\partial s^2} - \frac{\partial^2 x_2}{\partial s^2}\frac{\partial y_1}{\partial t}}. \tag{33}$$

Equation (33) implies that we can compute the first derivative $(C_1 * C_2)'(\bar{t})$, for some $\bar{t} \in [t_0, t_1]$, once we can find the corresponding parameter $\bar{s} \in [s_0, s_1]$ such that $C_1'(\bar{t}) \parallel C_2'(\bar{s})$, even without knowing the exact reparametrizing function $s(t)$.

Because $C_1(t)$ and $C_2(s)$ are compatible with each other, we have $C_1'(t_0) \parallel C_2'(s_0)$ and $C_1'(t_1) \parallel C_2'(s_1)$; namely, we have two exact tangent vectors at the two end points of the convolution curve segment. An approximated convolution curve can be computed using the cubic Hermite interpolation of the two end points and the two tangent vectors:

$$\begin{aligned}
(C_1 * C_2)(t_0) &= C_1(t_0) + C_2(s_0), \\
(C_1 * C_2)(t_1) &= C_1(t_1) + C_2(s_1), \\
\frac{\partial (C_1 * C_2)}{\partial t}(t_0) &= \frac{\partial C_1}{\partial t}(t_0) + \frac{\partial C_2}{\partial s}(s_0)\frac{\partial s}{\partial t}(t_0), \\
\frac{\partial (C_1 * C_2)}{\partial t}(t_1) &= \frac{\partial C_1}{\partial t}(t_1) + \frac{\partial C_2}{\partial s}(s_1)\frac{\partial s}{\partial t}(t_1),
\end{aligned}$$

where $(\partial s/\partial t)(t_0)$ and $(\partial s/\partial t)(t_1)$ are computed using Equation (33). Figure 22 shows an example of HIC approximation curve.

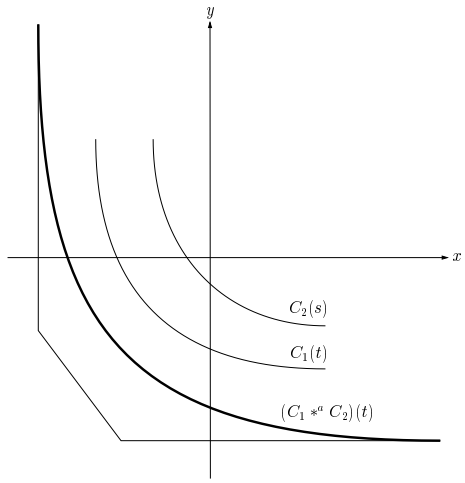


Figure 22: Approximated convolution curve of the Hermite interpolation method

4.3 Quadratic Curve Approximation Based Method (QAC)

Assume that $C_2(s)$ is approximated by a quadratic curve segment $Q_2(s)$. A quadratic curve $Q_2(s)$ has a linear hodograph (derivative curve) $Q_2'(s) = (as + b, cs + d)$, $s_0 \leq s \leq s_1$, where $a, b, c, d \in R$. ¿From the parallel relation, $Q_2'(s(t)) \parallel C_1'(t)$, that is:

$$(as(t) + b, cs(t) + d) \parallel (x_1'(t), y_1'(t)), \tag{34}$$

we have

$$\frac{cs(t) + d}{as(t) + b} = \frac{y_1'(t)}{x_1'(t)}. \tag{35}$$

Consequently, we have a reparametrization function $s(t)$ as follows:

$$s(t) = \frac{by_1'(t) - dx_1'(t)}{cx_1'(t) - ay_1'(t)}. \tag{36}$$

Then the approximated convolution curve is defined by:

$$(C_1 *^a C_2)(t) = C_1(t) + Q_2(s(t)), \quad t_0 \leq t \leq t_1, \tag{37}$$

where $s(t)$ is given in Equation (36). For a polynomial curve $C_1(t)$ of degree d, the reparametrized curve $Q_2(s(t))$ is a planar rational curve of degree $2(d-1)$. Thus, the approximated convolution curve, $(C_1 *^a C_2)(t)$ is a planar rational curve of degree $3d - 2$. For a rational curve $C_1(t)$ of degree d, the function $s(t)$ is a rational polynomial of degree $2d - 2$. (Note that the highest degree terms both in the numerator and the denominator are canceled.) Therefore, $Q_2(s(t))$ is a rational curve of degree $2(2d - 2)$, and $(C_1 *^a C_2)(t)$ is a rational curve of degree $5d - 4$.

The quadratic Bézier curve approximation $Q_2(s)$ (Figure 23) has three control points, P_1, P_2, and P_3; thus the curve equation of $Q_2(s)$ is given by:

$$Q_2(s) = (1 - s)^2 P_0 + 2s(1 - s)P_1 + s^2 P_2, \; s_0 \leq s \leq s_1. \tag{38}$$

The simplest construction of $Q_2(s)$ is based on: (i) identifying the two end points of $Q_2(s)$ with the two end points of $C_2(s)$, i.e., $C_2(s_0) = Q_2(s_0) = P_0$, and $C_2(s_1) = Q_2(s_1) = P_2$, and (ii) setting the middle control point P_1 as the intersection point between the two tangent lines of $C_2(s)$ at s_0 and s_1, respectively. This simple approximation method guarantees the G^1-continuity between any two

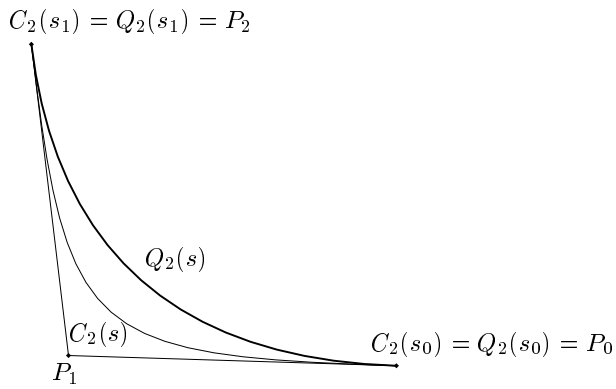


Figure 23: Approximation of $C_2(s)$ by quadratic Bézier curve $Q_2(s)$ (bold curve)

consecutive quadratic approximation curve segments, when the original curve segments are connected with G^1-continuity.

The approximation error of $(C_1 *^a C_2)(t)$ in Equation (37) is bounded by the approximation error of $Q_2(s)$. The approximation error between $C_2(s)$ and $Q_2(s) = (x_q(s), y_q(s))$ can be estimated with a distance function. The distance between $C_2(s)$ and $Q_2(s)$ is bounded by the maximum of the following error function:

$$\epsilon(s) = ||C_2(s) - Q_2(s)|| = \sqrt{(x_2(s) - x_q(s))^2 + (y_2(s) - y_q(s))^2}. \tag{39}$$

Instead of using $\epsilon(s)$ which has a square root term, we may use the following function $\varepsilon(s)$ for the error estimation:

$$\varepsilon(s) = ||C_2(s) - Q_2(s)||^2, \tag{40}$$

which can be computed using symbolic and numeric computation tools [8].

Algorithm 4.1 shows a divide-and-conquer algorithm for the computation of QAC. (See Elber and Cohen [7] for a similar algorithm that approximates an offset curve.) For the sake of simplicity, we assume that the two input curves have the same convexity (see Definition 3.2). When both $C_2(s)$ and $Q_2(s)$ are polynomial/rational curves, the error functional $\varepsilon(s)$ in Line (1) of Algorithm 4.1 is a polynomial/rational function. Because of the convex hull property of $\varepsilon(s)$, we can easily bound $\max(\varepsilon(s))$ by scanning for the largest coefficient of its control polygon. Moreover, the node parameter value of the control point (with the largest coefficient) may be used as the subdivision parameter for Line (2) of Algorithm 4.1. In Line (3), we use a numeric computation function to refine the parameter t_m.

Figure 24 shows the QAC approximation of two cubic B-spline curves. $C_1(t)$ and $C_2(s)$ in Figure 24(a) are cubic B-spline curves with five and twenty seven control points, respectively. After the compatible subdivision, we compute the QAC approximation curves with various tolerance values of approximation error, ϵ. Figures 24 (c)–(f) show the QAC approximations and the numbers of their control points. The QAC approximations are piecewise rational B-spline curves of degree seven. In Figure 24(b), the trace of the QAC approximation curve (bold solid curve) is verified by sweeping $C_2(s)$ (a family of light curves) along $C_1(t)$ (dashed curve).

When we approximate $C_1(t)$ with a quadratic curve $Q_1(t)$ (as well as approximating $C_2(s)$ with $Q_2(s)$) and compute $s(t)$ using $Q_1'(t)$, QAC approximation is a rational curve of degree four. Figure 25 shows the degree and the number of control points of various QAC approximations computed by the following four methods:

Algorithm 4.1

Input:

$C_1(t) = (x_1(t), y_1(t))$, $t_0 \le t \le t_1$, and

$C_2(s) = (x_2(s), y_2(s))$, $s_0 \le s \le s_1$: two compatible regular freeform curves;

ϵ: maximal tolerance of approximation;

Output:

$(C_1 *^a C_2)(t)$, $t_0 \le t \le t_1$: an approximated convolution of $C_1(t)$ and $C_2(s)$;

Algorithm: **QAC**$(C_1(t),\ C_2(s),\ \epsilon)$

```
begin
    Q_2(s) ⇐ quadratic approximation of C_2(s);
(1) if √(max ||C_2(s) − Q_2(s)||^2) < ε then begin
        (as + b, cs + d) ⇐ Q_2'(s);
        s(t) ⇐ (b y_1'(t) − d x_1'(t)) / (c x_1'(t) − a y_1'(t));
        return C_1(t) + Q_2(s(t));
    end
    else begin
(2)     s_m ⇐ parameter of C_2(s), s_0 ≤ s_m ≤ s_1,
              where ||C_2(s) − Q_2(s)||^2 has a maximum value;
(3)     Compute t_m such that ⟨C_1'(t_m), N_2(s_m)⟩ = 0;
        C_{1,1}(t), C_{1,2}(t) ⇐ subdivide C_1(t) at t_m;
        C_{2,1}(s), C_{2,2}(s) ⇐ subdivide C_2(s) at s_m;
        return MergeCurves(QAC(C_{1,1}(t), C_{2,1}(s), ε), QAC(C_{1,2}(t), C_{2,2}(s), ε));
    end
end
```

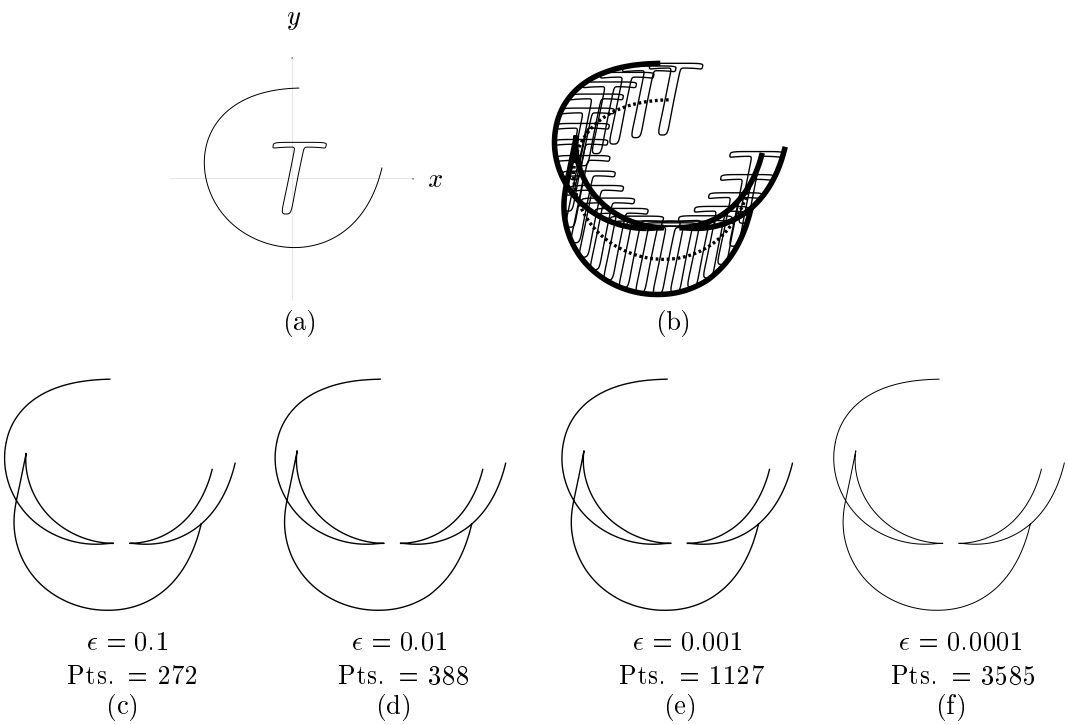
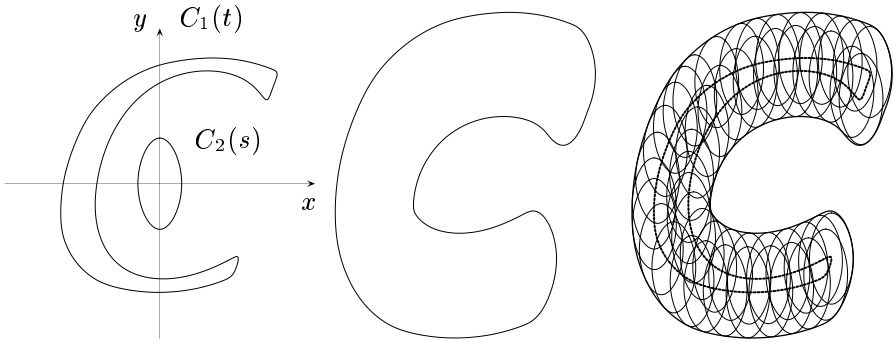


Figure 24: QAC approximation of two cubic B-spline curves

- $C_1(t) + Q_2(s(t))$: approximate only $C_2(s)$ with $Q_2(s)$. $s(t)$ is computed using $C_1'(t)$.
- $C_2(s) + Q_1(t(s))$: approximate only $C_1(t)$ with $Q_1(t)$. $t(s)$ is computed using $C_2'(s)$.



(a) (b) (c)

QAC	Degree	Number of control points		
		$\epsilon = 0.1$	$\epsilon = 0.01$	$\epsilon = 0.001$
$C_1(t) + Q_2(s(t))$	rational 7	550	638	953
$C_2(s) + Q_1(t(s))$	rational 11	464	764	2240
$Q_1(t) + Q_2(s(t))$	rational 4	188	333	989
$Q_2(s) + Q_1(t(s))$	rational 4	182	329	949

Figure 25: Various QAC approximations computed from cubic $C_1(t)$ and rational cubic $C_2(s)$

- $Q_1(t) + Q_2(s(t))$: approximate both $C_1(t)$ and $C_2(s)$ with $Q_1(t)$ and $Q_2(s)$, respectively. $s(t)$ is computed using $Q_1'(t)$.
- $Q_2(s) + Q_1(t(s))$: approximate both $C_1(t)$ and $C_2(s)$ with $Q_1(t)$ and $Q_2(s)$, respectively. $t(s)$ is computed using $Q_2'(s)$.

Note that the QAC approximations shown in Figure 25 are computed from a cubic B-spline curve $C_1(t)$ and a rational cubic B-spline curve $C_2(s)$. Figure 25(c) verifies the convolution curve by sweeping $C_2(s)$ (a family of light curves) along $C_1(t)$ (dashed curve).

4.4 Reparametrization Based Methods

Another natural approach to the approximation of a convolution curve is to approximate the reparametrization function $s(t)$ in Equation (2) using a polynomial/rational function, rather than approximating the whole convolution curve $(C_1 * C_2)(t)$. In this section, we present three such methods.

4.4.1 Convolution Using Linear Reparametrization (LRC)

The simplest approximation of the reparametrization $s(t)$ is a simple translation and scaling of the parameter domain $[s_0, s_1]$ to $[t_0, t_1]$, that is:

$$s(t) = s_0 + \frac{t - t_0}{t_1 - t_0}(s_1 - s_0). \tag{41}$$

We call this method LRC (Linear Reparametrization Convolution). The implementation of LRC requires a simple linear reparametrization of $s(t)$ and the addition of two curve segments $C_1(t)+C_2(s(t))$. Figure 26 shows an example of LRC approximation.

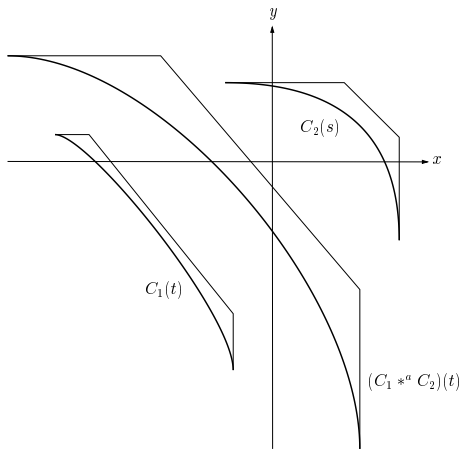


Figure 26: LRC approximation $(C_1 *^a C_2)(t)$ of $C_1(t)$ and $C_2(s)$.

4.4.2 Convolution Using Tangent Field Matching (TMC)

We may approximate the reparametrization $s(t)$ using the technique of *tangent field matching* suggested in Cohen et al. [6]. The basic concept of this method is explained below.

Given a freeform curve, its specific parametrization has an important implication for the processing of the curve. The arc-length parametrization provides many useful properties. For example, in computer animation, motion speed control becomes much easier when we have an almost arc-length parametrization of the motion curve. In Computer Aided Geometric Design, the major concern is how to design various geometric shapes using freeform curves and surfaces. In this case, the curve and surface traces are more important than the parametrization itself. However, further geometric processings on these freeform shapes are heavily dependent on the parametrization of the curves and surfaces. Proper reparametrization can also improve the rendering quality of freeform curves and surfaces significantly, when the freeform objects are rendered with polygonal approximation. Kosters [28] used a curvature dependent parametrization to render the freeform curves and surfaces with more line segments in high curvature regions. In many geometric operations on two operand curves, their *tangent field matching* plays an important role. We provide a formal definition of this concept as follows:

Definition 4.1 *Let $C_1(u)$, $u_0 \le u \le u_1$, and $C_2(v)$, $v_0 \le v \le v_1$, be two regular C^1 parametric curves. Consider two reparametrizations, $U : u \mapsto t$ and $V : v \mapsto t$, which map $C_1(u)$ into the curve $\hat{C}_1(t)$, $t_0 \le t \le t_1$, and $C_2(v)$ into the curve $\hat{C}_2(t)$, $t_0 \le t \le t_1$, respectively. If two unit tangent fields:*

$$T_1(t) = \frac{\hat{C}_1'(t)}{||\hat{C}_1'(t)||} \quad \textit{and} \quad T_2(t) = \frac{\hat{C}_2'(t)}{||\hat{C}_2'(t)||}, \tag{42}$$

are the same for all $t \in [t_0, t_1]$, the two parametrized curves $\hat{C}_1(t)$ and $\hat{C}_2(t)$ have a complete tangent field matching, *and the two tangent fields of $C_1(u)$ and $C_2(v)$ are* matched by the reparametrizations U and V. □

When the two curves have a complete tangent field matching, the inner product $\langle T_1(t), T_2(t) \rangle = 1$, for all $t \in [t_0, t_1]$. Cohen et al. [6] presented an algorithm to approximate the matching by solving the following optimization problem:

$$\max_{v(u)} \int_{u_0}^{u_1} \left\langle \frac{C_1'(u)}{||C_1'(u)||}, \frac{C_2'(v(u))}{||C_2'(v(u))||} \right\rangle du, \tag{43}$$

where $v(u) = V(U^{-1}(u))$, $v(u_0) = v_0$, and $v(u_1) = v_1$. This optimization problem is bounded from above by $u_1 - u_0$, since the normalized inner product does not exceed one. While the solution of Equation (43) is difficult, we can solve instead an associated discrete optimization problem which may produce an arbitrarily close approximation to the solution of Equation (43). We sample both $C_1(u)$ and $C_2(v)$ at n uniform parameter locations and compute their unit tangents as $T_{1,i}$ and $T_{2,j}$, $0 \leq i, j < n$. Then, the problem is reduced to a discrete optimization problem:

$$\max_{j(i)} \sum_{i=0}^{n-1} \left\langle T_{1,i}, T_{2,j(i)} \right\rangle \tag{44}$$

subject to:

$$j(0) = 0, \qquad j(n-1) = n-1, \qquad j(i) \leq j(i+1). \tag{45}$$

Cohen et al. [6] suggested a method to solve the optimization problem of Equation (44) within $O(n^2)$ time, where n is the number of sample locations. This method employs a dynamic programming technique and provides a globally optimal solution. The more sample points we use, the closer is the resulting reparametrized curve $C_2(v(u))$ to the completely matching curve with $C_1(u)$. In Reference [6], a method is also described to approximate a continuous function $v(u)$ from the discrete match of $j(i)$, by using a least squares fitting to a B-spline curve. The composition of $C_2(v(u))$ can then be computed using symbolic computation tools [8, 26], while resulting in a B-spline curve representation. The tangent field matching can be used for various practical geometric operations. Cohen et al. [6] suggested efficient algorithms which can prevent self-intersections in the construction of ruled surfaces, blending surfaces, sweep surfaces, and metamorphosis between two parametric curves.

Tangent field matching allows the computation of an approximated convolution between $C_1(t)$ and $C_2(s)$, by first computing the proper reparametrization $s(t)$ for $C_2(s(t)) = (x_2(s(t)), y_2(s(t)))$. Unfortunately, the approximation error cannot be computed with the distance function which we used in the quadratic curve approximation (see Section 5.2.1). Instead of the distance function, we use the following formula which represents the value of $\cos^2 \alpha$, where α is the angle between the tangent vector of $C_1(t)$ and the normal vector of $C_2(s(t))$:

$$\delta(t) = \frac{\left\langle C_1'(t), \bar{N}_2(s(t)) \right\rangle^2}{||C_1'(t)||^2 ||\bar{N}_2(s(t))||^2}, \tag{46}$$

where $\bar{N}_2(s(t)) = (y_2'(s(t)), -x_2'(s(t)))$ is an unnormalized normal vector field of $C_2(s(t))$. Then, the angle deviation can be represented as $\epsilon = \left\| \frac{\pi}{2} - \arccos(\sqrt{\max \delta(t)}) \right\|$.

Algorithm 4.2 is an iterative algorithm for computing the TMC approximation. The accuracy of the matching algorithm [6] is controlled by the sampling value of $N = N_0$. According to experimental results, $N_0 = 15$ has been found to be a reasonable starting value for computing $j(i)$ for a single cubic polynomial curve segment with no inflection point. As in the case of QAC, $\max(\delta(t))$ in Line (2) of Algorithm 4.2 can be found by scanning the control polygon of $\delta(t)$. Figure 27 shows the TMC approximations computed from two cubic B-spline curves with six and ten control points, respectively. All the TMC approximations of Figure 27 are piecewise cubic B-spline curves, while using a linear reparametrization $s(t)$. Note that the open B-spline curve has a curve direction from left to right. In Figure 27(b), the upper envelope curves do not contribute to the convolution curve since they are generated by the closed B-spline curve points, at which the curve tangent directions are opposite to those of the open B-spline curve.

4.4.3 Convolution Using Sample Reparametrization (SRC)

Instead of approximating $s(t)$ with tangent field matching, we may approximate $s(t)$ by a simpler method. For each sample parameter value $\bar{t}_i \in [t_0, t_1]$ of $C_1(t)$, the corresponding $\bar{s}_i \in [s_0, s_1]$ of $C_2(s)$

Algorithm 4.2

Input:

$C_1(t) = (x_1(t), y_1(t))$, $t_0 \le t \le t_1$, and

$C_2(s) = (x_2(s), y_2(s))$, $s_0 \le s \le s_1$: two compatible regular freeform curves;

Δ: maximal tolerance of angle deviation in approximation;

Output:

$(C_1 *^a C_2)(t)$, $t_0 \le t \le t_1$: a TMC approximation of $C_1(t)$ and $C_2(s)$;

Algorithm: TMC($C_1(t)$, $C_2(s)$, Δ)

```
begin
```

$N \Leftarrow N_0$;

```
    do
```

(1) $C_2(s(t)) \Leftarrow$ **TangentFieldMatching**($C_1(t)$, $C_2(s)$, N);

$N_2(s(t)) \Leftarrow (y_2'(s(t)), -x_2'(s(t)))$;

$\delta(t) \Leftarrow \frac{\left\langle C_1'(t), N_2(s(t)) \right\rangle^2}{||C_1'(t)||^2 \, ||N_2(s(t))||^2}$;

$N \Leftarrow 2N$;

(2) while ($\left\| \frac{\pi}{2} - \arccos(\sqrt{\max(\delta(t))} \right\| > \Delta$)

```
    return C1(t) + C2(s(t));
end
```

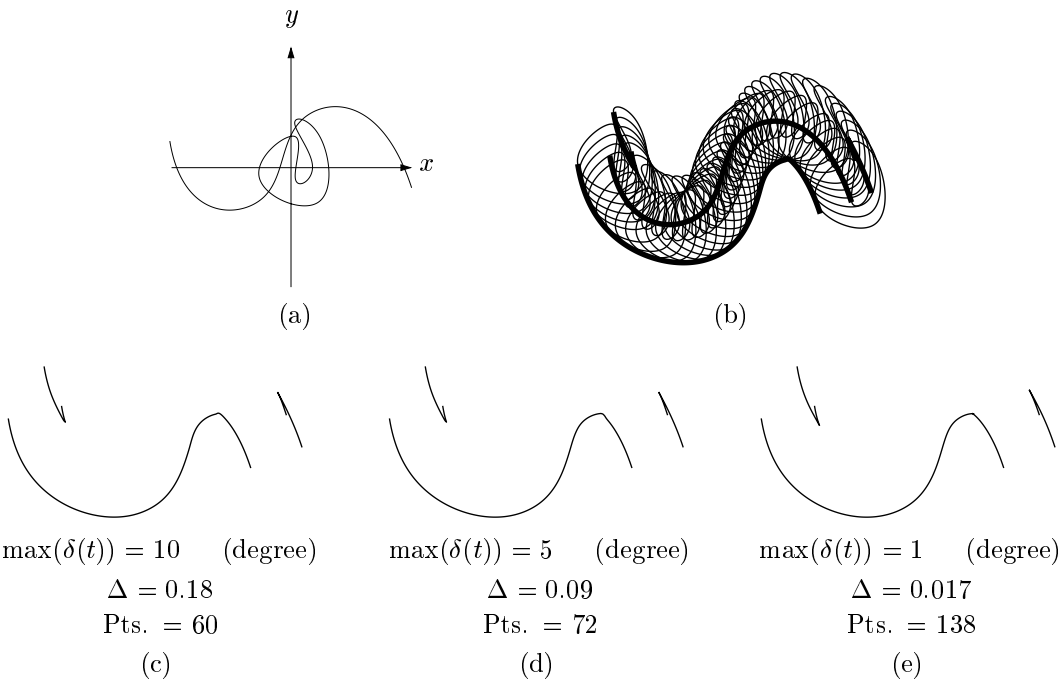


Figure 27: TMC approximation of two cubic B-spline curves

can be computed by solving Equation (27). Using these sample parameter values, the reparametrization $s(t)$, $t_0 \le t \le t_1$, such that $s(\bar{t}_i) = \bar{s}_i$, can be approximated or interpolated using the same techniques for the LSC and BIC methods. Let N be the number of sample parameter values. The simplest linear $s(t)$ in uniform B-spline representation has N control points:

$$\{\bar{s}_i \mid 0 \le i < N\}, \tag{47}$$

and the following knot vector:

$$\{\bar{t}_0, \bar{t}_0, \bar{t}_1, \bar{t}_2, ..., \bar{t}_{N-2}, \bar{t}_{N-1}, \bar{t}_{N-1}\}. \tag{48}$$

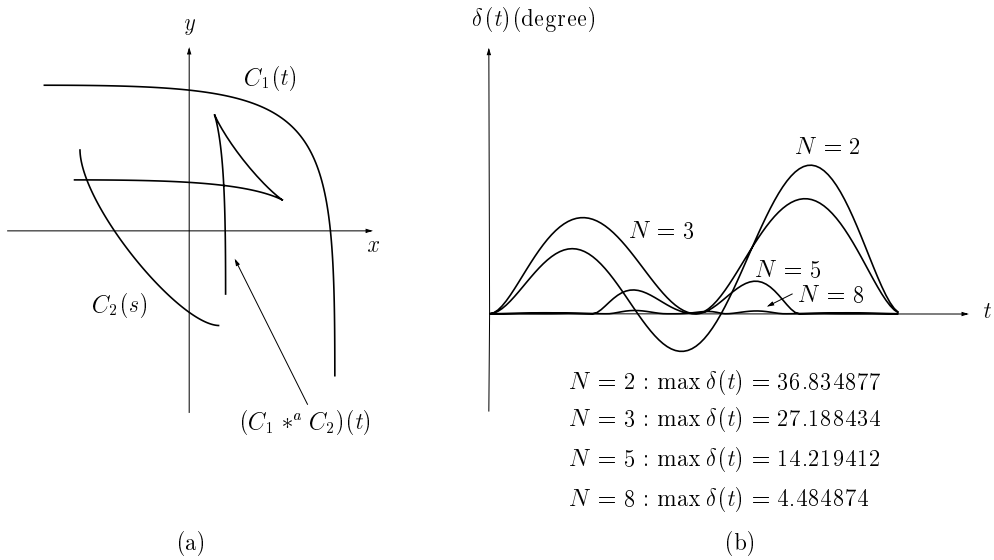


Figure 28: SRC approximation for two planar curves: (a) $C_1(t)$, $C_2(s)$, and $(C_1 *^a C_2)(t)$, (b) Error function $\delta(t)$ for various N (the number of sample parameters)

This method, called SRC (Sample Reparametrization Convolution), is much simpler than the tangent field matching procedure of TMC. In Algorithm 4.2, we can replace the procedure **TangentField-Matching** by the code segment described in Algorithm 4.3. Figure 28 shows an example of the SRC approximation and the refinement procedure of the error function $\delta(t)$ by increasing the number of sample parameters.

Algorithm 4.3

$\{\bar{t}_i\}$, $0 \leq i < N \Leftarrow N$ parameter values uniformly sampled in $[t_0, t_1]$;
for each $i = 0, 1, ..., N-1$ do
    Compute $\bar{s}_i$ such that $\langle C_2'(\bar{s}_i), N_1(\bar{t}_i) \rangle = 0$;
$s(t) \Leftarrow$ reparametrization defined by Equations (47) and (48);
$C_2(s(t)) \Leftarrow$ symbolic composition of $C_2(s)$ and $s(t)$;

5 Comparisons of Convolution Curve Approximation Methods

In Section 4, we presented several methods for convolution curve approximation. In this section, we report various results of comparing these approximation methods.

5.1 Qualitative Comparisons

Each approximation method presented in the previous section has advantages and disadvantages. In terms of complexity of computation and implementation, LRC is the simplest method. LSC, BIC, and TMC require more computation time due to the intermediate steps such as least squares approximation, interpolation, and tangent field matching.

QAC is the only method in which we can guarantee that the resulting approximated convolution curve is within the ϵ–band from the exact convolution curve, where ϵ is the given tolerance of quadratic curve approximation. (In fact, this statement is true only for non-trimmed (closed or infinite) input curves and for the convolution curves after eliminating the self-intersection loops; see Reference [30]

for more details.) As we will describe in Section 5.2, other methods do not guarantee that the approximated convolution curve is within a certain given distance from the exact convolution curve. Furthermore, in the QAC method, the approximation error can be estimated in an *a priori* fashion; that is, we can estimate the convolution curve approximation error by measuring the approximation error of quadratic curve approximation, even without computing any approximated convolution curve at all. In other methods, the approximation error can be estimated only after an approximated convolution curve is constructed (see Section 5.2). In a subdivision-based algorithm (Algorithm 5.1), the approximation error estimation is required at each subdivision step. For some input curves of special types (e.g., circles), the quadratic curve approximation error is already known. In that case, QAC saves computation time since there is no need to estimate the convolution curve approximation error at each subdivision step.

In some approximation methods such as LSC and BIC, we can easily control the degree of an approximated convolution curve. The HIC method produces cubic curves only. In CTC, QAC, and LRC methods, the degree of an approximated convolution curve depends on the degrees of two input curves; thus we cannot control the degree of a convolution curve arbitrarily. Moreover, the QAC method always generates rational curves. The degree of a convolution curve generated by TMC or SRC is mainly dependent on the degree of the reparametrization $s(t)$. When two input curves are both polynomial curves, TMC and SRC generate lower degree convolution curves than QAC. Note that the reparametrized curve $C_2(s(t))$ in TMC or SRC has the same degree as $C_2(s)$ after composition when we use a linear reparametrization function $s(t)$. Table 1 compares the degrees of approximated convolution curves generated by different methods. Note that we do not consider variants of QAC (see Figure 25). We also use a linear reparametrization $s(t)$ for TMC and SRC.

C_1	C_2	CTC	LSC, BIC	HIC	QAC	LRC	TMC, SRC
d_1	d_2	$\max(d_1, d_2)$	any	3	rt. $(3d_1 - 2)$	$\max(d_1, d_2)$	$\max(d_1, d_2)$
d_1	rt. d_2	rt. $\max(d_1, d_2)$	any	3	rt. $(3d_1 - 2)$	rt. $(d_1 + d_2)$	rt. $(d_1 + d_2)$
rt. d_1	d_2	rt. $\max(d_1, d_2)$	any	3	rt. $(5d_1 - 4)$	rt. $(d_1 + d_2)$	rt. $(d_1 + d_2)$
rt. d_1	rt. d_2	rt. $\max(d_1, d_2)$	any	3	rt. $(5d_1 - 4)$	rt. $(d_1 + d_2)$	rt. $(d_1 + d_2)$

Table 1: Degrees of various convolution approximations. "rt." represents rational curve.

5.2 Quantitative Comparisons

5.2.1 Error Estimation

The distance between the exact and approximated convolution curves is given by

$$\begin{aligned} &\|(C_1 * C_2)(t) - (C_1 *^a C_2)(t)\| \\ &\quad = \|C_1(t) + C_2(s(t)) - (C_1(t) + C_2(s^a(t)))\| \\ &\quad = \|C_2(s(t)) - C_2(s^a(t))\|, \end{aligned} \tag{49}$$

where $s^a(t)$ is an approximation of the exact reparametrization $s(t)$. The main difficulty in measuring the convolution approximation error is that we do not have the exact reparametrization $s(t)$, in any convolution approximation method we have considered in this paper. Furthermore, in the control polygon and interpolation based approaches, we do not even have the approximated reparameterization $s^a(t)$.

In this subsection, we suggest two different criteria which can be applied to divide-and-conquer algorithms. In Section 5.2.3, these error estimation functions will be used in quantitative comparisons of various convolution curve approximation methods.

Distance Sampling: ¿From Equation (2), we have

$$(C_1 * C_2)(t) - C_1(t) = C_2(s(t)). \tag{50}$$

Note that $C_2(s(t))$ has the same curve trace as that of $C_2(s)$, where $s(t)$ is an exact reparametrization. Let

$$\tilde{C}_2(t) = (C_1 *^a C_2)(t) - C_1(t). \tag{51}$$

We can use the Hausdorff distance between $\tilde{C}_2(t)$ and $C_2(s)$ to estimate the approximation error. However, the relationship between the two parameters $t \in [t_0, t_1]$ and $s \in [s_0, s_1]$ of $\tilde{C}_2(t)$ and $C_2(s)$, respectively, is not well-defined for most approximation methods such as the control polygon and interpolation based approaches. Although distance sampling does not guarantee the maximum global error [8], this seems the only available method that can measure the maximum distance between $\tilde{C}_2(t)$ and $C_2(s)$.

Let $\tilde{C}_2(\bar{t}_i)$ and $C_2(\bar{s}_i)$, $(i = 0, 1, ..., n-1)$, be n finite sample points on the curves $\tilde{C}_2(t)$ and $C_2(s)$, respectively, where $\bar{t}_0 = t_0$, $\bar{t}_{n-1} = t_1$, $\bar{s}_0 = s_0$, and $\bar{s}_{n-1} = s_1$. The distance from a point $\tilde{C}_2(\bar{t}_i)$ to the curve $C_2(s)$ is approximated by

$$\min_{j=0}^{n-1} \|\tilde{C}_2(\bar{t}_i) - C_2(\bar{s}_j)\|, \tag{52}$$

and the maximum distance between $\tilde{C}_2(t)$ and $C_2(s)$ is approximated by

$$\max_i \left\{ \min_j \|\tilde{C}_2(\bar{t}_i) - C_2(\bar{s}_j)\| \right\}. \tag{53}$$

Note that the distance sampling function (Equation (53)) cannot be used in the LRC, TMC and SRC method. In these three methods, the reparametrization $s(t)$ is approximated while maintaining the same trace with $C_2(s)$. Thus, the distance $\|\tilde{C}_2(t) - C_2(s)\|$ always vanishes.

Normal Deviation: Another criterion to measure the error in each convolution curve approximation method is to compare the normal directions of $\tilde{C}_2(t)$ and $C_1(t)$. For the exact convolution computation, $\tilde{C}_2'(t) \parallel C_1'(t)$. The normal vector deviation between $\tilde{C}_2(t)$ and $C_1(t)$ can be represented by the following equation:

$$\delta(t) = \frac{\left\langle C_1'(t), \tilde{N}_2(t) \right\rangle^2}{\|C_1'(t)\|^2 \|\tilde{N}_2(t)\|^2}, \tag{54}$$

where $\tilde{N}_2(t)$ is an unnormalized normal vector field of $\tilde{C}_2(t)$. The angle between $N_1(t)$ and $\tilde{N}_2(t)$ is measured by

$$\left\| \frac{\pi}{2} - \arccos(\sqrt{\max \delta(t)}) \right\|, \tag{55}$$

as mentioned in Section 4.4.2. Note that Equation (54) is the same as Equation (46) in the TMC approximation method.

Nevertheless, for the QAC method, we cannot use the normal deviation function to estimate the approximation error. In the QAC method, $(C_1 *^a C_2)'(t)$ is always parallel to $C_1'(t)$, because

$$(C_1 *^a C_2)(t) = C_1(t) + Q_2(s(t)), \tag{56}$$

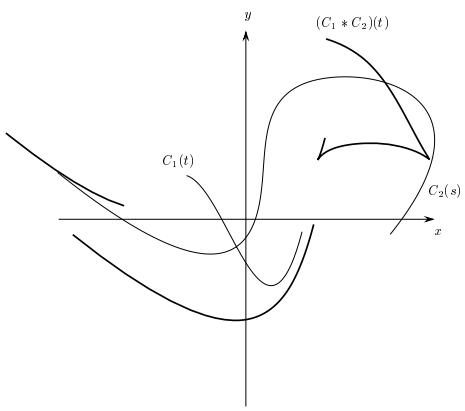
where $C_1'(t) \parallel Q_2'(s(t))$. Thus, $\tilde{C}_2'(t) = Q_2'(s(t))$ is also parallel to $C_1'(t)$. In other words, QAC preserves the exact normal direction, while generating distance deviation.

Algorithm 5.1
Input:
$C_1(t) = (x_1(t), y_1(t))$, $t_0 \le t \le t_1$, and
$C_2(s) = (x_2(s), y_2(s))$, $s_0 \le s \le s_1$: two regular compatible freeform curves;
ϵ: maximal tolerance of approximation;
Output:
$(C_1 *^a C_2)(t)$, $t_0 \le t \le t_1$: an approximated convolution of $C_1(t)$ and $C_2(s)$;
Algorithm: SubdivConvolution $(C_1(t),\ C_2(s),\ \epsilon)$
begin
$(C_1 * C_2)^a(t) \Leftarrow$ an approximated convolution curve;
$\epsilon_* \Leftarrow$ distance computed by sampling, or normal deviation;
(1) if $\epsilon_* < \epsilon$ then return $(C_1 * C_2)^a(t)$;
else begin
(2) $t_m \Leftarrow (t_0 + t_1)/2$;
Compute s_m such that $\langle C_2'(s_m), N_1'(t_m)\rangle = 0$;
$C_{1,1}(t),\ C_{1,2}(t) \Leftarrow$ subdivide $C_1(t)$ at t_m;
$C_{2,1}(s),\ C_{2,2}(s) \Leftarrow$ subdivide $C_2(s)$ at s_m;
return **MergeCurves**(**SubdivConvolution**$(C_{1,1}(t), C_{2,1}(s), \epsilon)$,
SubdivConvolution$(C_{1,2}(t), C_{2,2}(s), \epsilon)$);
end
end

5.2.2 Subdivision Based Algorithm

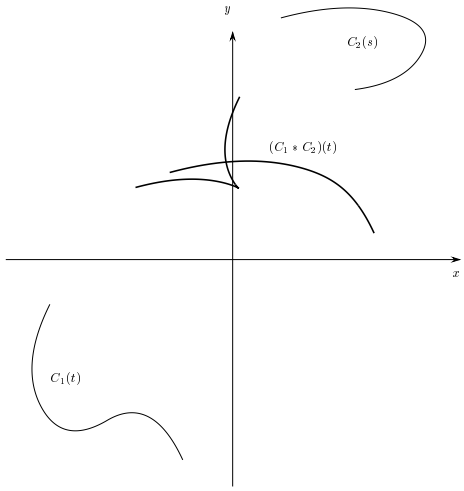
A general subdivision based algorithm for convolution curve approximation is described in Algorithm 5.1, where an appropriate error estimation function is assumed. In Line (1), we apply a distance sampling function or a normal deviation function to compute the error in the convolution curve approximation. Instead of using the naive bisection method as shown in Line (2), we can subdivide the parameter domain at the parameter value corresponding to the maximum of an error function computed from Equation (53) or Equation (55).

5.2.3 Comparison Results



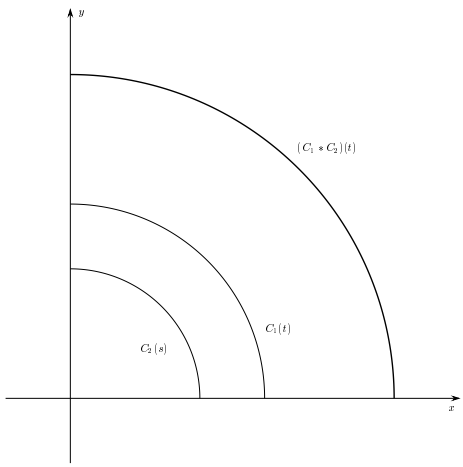
ϵ_d	CTC	LSC	BIC	HIC	QAC	ϵ_n	CTC	LSC	BIC	HIC	LRC	TMC	SRC
1.0	31	31	31	31	71	10°	43	40	40	46	41	40	40
0.1	34	31	31	31	71	5°	55	43	43	58	50	49	49
0.01	70	34	35	31	71	3°	97	45	44	64	65	61	61
0.001	193	47	51	79	92	1°	244	54	59	118	104	94	106
						0.5°	463	62	61	166	143	118	145

Figure 29: Convolution curve approximation of two cubic B-spline curves.



ϵ_d	CTC	LSC	BIC	HIC	QAC	ϵ_n	CTC	LSC	BIC	HIC	LRC	TMC	SRC
1.0	12	12	12	15	24	10°	24	17	19	22	18	22	20
0.1	18	13	14	15	24	5°	36	21	21	28	22	28	22
0.01	36	18	18	24	50	3°	54	24	24	34	24	32	26
0.001	108	31	33	36	56	1°	166	35	34	49	40	46	38
						0.5°	330	43	44	76	54	62	46

Figure 30: Convolution curve approximation of two quadratic B-spline curves.

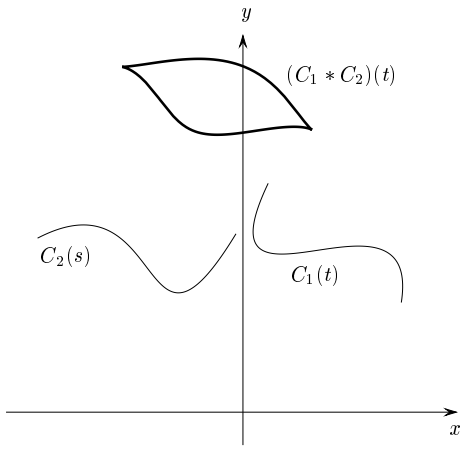


ϵ_d	CTC	LSC	BIC	HIC	QAC	ϵ_n	CTC	LSC	BIC	HIC	LRC	TMC	SRC
1.0	3	3	3	4	7	10°	9	4	3	7	3	3	3
0.1	5	3	3	7	7	5°	17	4	4	7	3	3	3
0.01	13	5	5	7	21	3°	17	6	6	13	3	3	3
0.001	33	8	9	13	35	1°	65	8	9	13	3	3	3
0.0001	129	16	17	25	56	0.5°	129	10	11	19	3	3	3

Figure 31: Convolution curve approximation of two exact circular arcs (rational quadratic B-spline curves).

Figures 29–32 show the results of quantitative comparison among different convolution curve approximation methods in terms of the tolerance of distance, ϵ_d, and the tolerance of normal deviation, ϵ_n, using Algorithm 5.1. The numbers in each table show the number of control points in each approximated convolution curve. Although the QAC method can compute the global approximation error representing the distance between the approximated and exact convolution curves, we apply the same distance sampling function to QAC for the sake of fair comparison with other methods.

In most of the test results, the LSC and BIC methods perform better than other methods. (Similar



ϵ_d	CTC	LSC	BIC	HIC	QAC	ϵ_n	CTC	LSC	BIC	HIC	LRC	TMC	SRC
1.0	13	13	13	13	29	10°	40	39	33	55	49	43	25
0.1	19	13	13	13	29	5°	49	42	37	73	58	49	43
0.01	52	27	32	46	57	3°	79	46	46	82	70	64	76
0.001	130	90	99	142	95	1°	196	98	99	136	121	106	208
						0.5°	376	179	182	196	178	175	442

Figure 32: Convolution curve approximation of two cubic Bézier curves.

results can be found in the comparison of offset curve approximation methods reported in Reference [10]). Next in performance rank is HIC, which is also an interpolation based method, followed by QAC, and the reparametrization based methods such as TMC and SRC. The control polygon based method, CTC, performs pretty badly, even worse than LRC. For the convolution of a circular arc (Figure 31), LRC, TMC, and SRC generate exact results. Although the table on the left-hand side of Figure 31 does not contain the results of LRC, TMC, and SRC, it is obvious that the LRC, TMC, and SRC methods produce exact results in terms of ϵ_d.

6 Elimination of Redundant Parts

In this section, we consider how to compute the Minkowski sum boundary of two planar curved objects. When two objects are convex, the Minkowski sum boundary is the same as the convolution curve of the two objects' boundary curves. However, for non-convex objects, the Minkowski sum boundary is a subset of the convolution curve.

For the construction of the Minkowski sum boundary, we first construct the convolution curve; after that, all local and global self-intersections are detected and redundant curve segments are eliminated. The determination of self-intersection loops is closely related to the arrangement of planar curve segments [19]. A robust implementation of curve arrangement is one of the most difficult open problems in geometric modeling. The only reliable robust implementation today involves using polygonal approximations of the convolution curve segments and determining the arrangement of resulting line segments. (See Guibas and Marimont [20] for the state-of-the-art of robust arrangement algorithms for line segments in the plane.)

Ahn et al. [1] demonstrated the efficiency and robustness of an arrangement technique for line segments in approximating the boundary of a 2D general sweep. General sweep is the most general form of sweep in which the moving object changes its shape dynamically while moving along a trajectory curve. Minkowski sum computation is a special case of general sweep computation. Therefore, the general technique of Ahn et al. [1] can be applied to the case of the Minkowski sum computation.

However, there are some computational shortcuts in the special case of the Minkowski sum computation [29]. In this section we consider other advantages in eliminating redundant convolution curve segments. We assume that the two planar curved objects are bounded by piecewise parametric curves. Note that in many applications of the Minkowski sum computation, we need to consider closed objects only.

With the exception of some obvious redundancies (to be discussed below), we approximate convolution curve segments by using discrete points and their connecting piecewise line segments (Figure 33(c)). In the next step, we use a plane sweep algorithm [34] to detect all the intersections among the convolution line segments (Figure 33(d)), and construct a polygonal approximation of the Minkowski sum boundary (Figure 33(e)).

Once we have computed a polygonal approximation of the Minkowski sum boundary, we can easily extract the parameter interval corresponding to each convolution curve segment on the Minkowski sum boundary. In particular, the coordinates and parameters corresponding to the self-intersections of the convolution curves (approximated by line segments) must be refined to more precise intersection points among exact convolution curve segments. For this purpose, the parameter values corresponding to each pair of intersecting line segments are used as an initial solution and numeric procedures are applied to improve the precision. Figure 33(f) shows the final Minkowski sum boundary thus constructed.

The true convolution curves are shown in Figure 33(b). Figure 33(c) shows polygonal approximations of some convolution curves (i.e., except some obviously redundant segments). One can easily notice that some curves of Figure 33(b) are missing in Figure 33(c) (in the polygonal approximation). To reduce the size of polygonal approximation data and to improve the robustness of line segment intersection in the plane sweep algorithm, we eliminate some obviously redundant edges from further consideration. The elimination procedure is based on the following three rules (see also Section 3.3):

- Rule 1: The convolution curves generated from two concave edges do not contribute to the final Minkowski sum boundary.
- Rule 2: The convolution curves generated from at least one concave vertex do not contribute to the final Minkowski sum boundary.
- Rule 3: The convolution curves that belong to a local self-intersection loop can be eliminated.

The elimination based on Rules 1 and 2 is explained in Section 3.3. In Rule 3, it is not easy to detect and eliminate all redundant convolution curve segments that belong to a local self-intersection loop. However, it is relatively easy to remove a certain portion of each self-intersection loop. Let $(C_1 * C_2)(t)$ be an exact convolution curve segment that is computed from two input curve segments, $C_1(t)$ and $C_2(t) = C_2(s(t))$, where $s(t)$ is a proper reparametrization. The convolution curve $(C_1 * C_2)(t)$ has a cusp at the parameter $\bar{t}$ such that

$$k_1(\bar{t}) = -k_2(\bar{t}), \tag{57}$$

that is, the curvature of $C_1(t)$ at $\bar{t}$ has the same magnitude as the curvature of $C_2(s(t))$ at $\bar{t}$, but with a different sign. In this case, $C_1(t)$ and $C_2(s)$ have different convexities.

When we subdivide the convolution curve $(C_1 * C)(t)$ at each cusp (equivalently, the compatible edges $C_1(t)$ and $C_2(s(t))$ at each $\bar{t}$ such that $k_1(\bar{t}) = -k_2(\bar{t})$), each resulting convolution curve segment $(C_1 * C_2)(t)$ is generated by a pair of convex-concave edges in which the concave edge has larger (respectively, smaller) curvature than the convex edge. The convolution curves generated by concave edges with larger curvature than the corresponding convex edges belong to redundant local self-intersection loops; thus they can be eliminated (see the part "A" in Figure 33(b)). All cusps can be approximated by computing the cusps of approximated convolution curves $(C_1 *^a C_2)(t)$.

Even after eliminating redundant convolution curve segments based on Rules 1–3, there are still many redundant segments in the planar graph of remaining convolution curves. Moreover, the elimination based on Rule 3 requires the construction of approximated convolution curves or the curvature

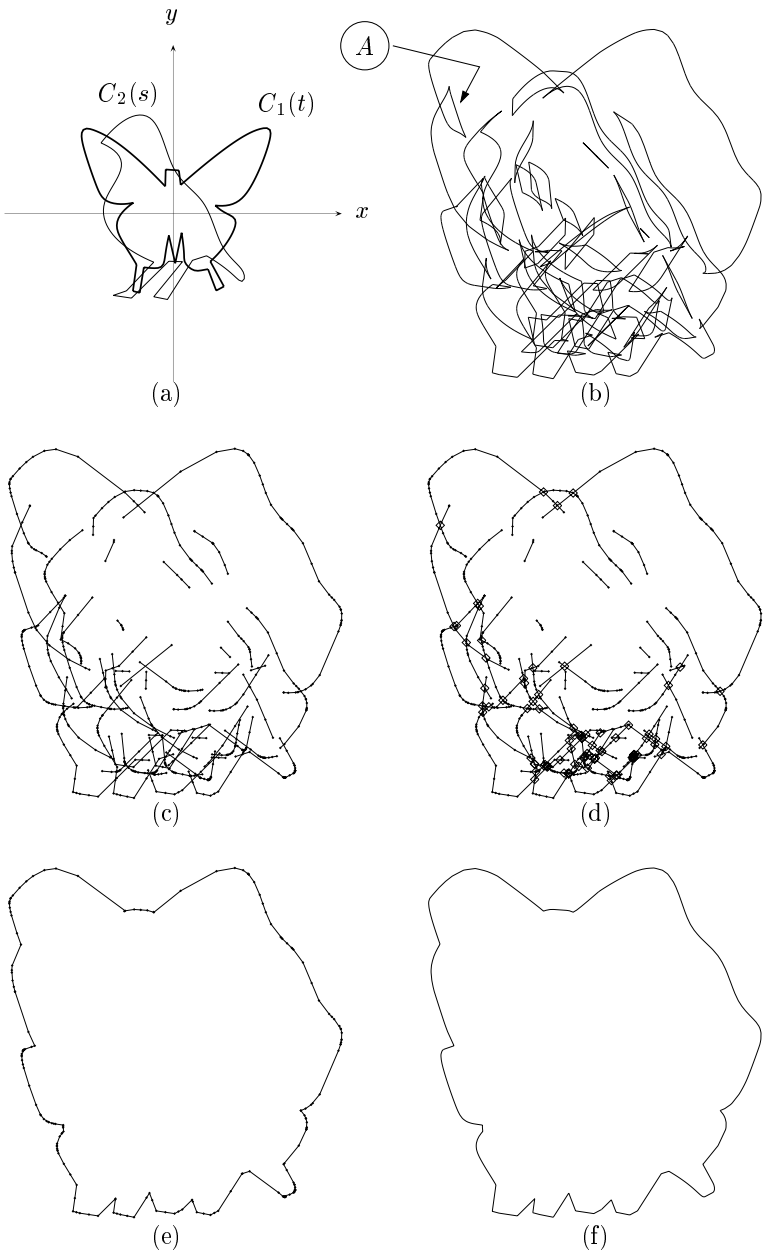


Figure 33: Computation steps for the elimination of self-intersection loops

comparison between two input curve segments. A more efficient solution is to generate a simpler Minkowski sum which is a proper subset of the Minkowski sum and then to eliminate convolution curve segments which appear in the interior of the simpler Minkowski sum.

Given two input objects O_1 and O_2, we approximate them with simpler proper subsets P_1 and P_2, respectively. Then, $P_1 \oplus P_2$ is also a proper subset of $O_1 \oplus O_2$. Figure 34 shows an example which uses rectilinear polygons P_i $(i = 1, 2)$. The resulting Minkowski sum $P_1 \oplus P_2$ is also a rectilinear polygon. The convolution curve segments of $C_1 * C_2$ that belong to the interior of $P_1 \oplus P_2$ can be eliminated from further consideration, where C_1 and C_2 are the boundary curves of O_1 and O_2, respectively. Figure 35 shows another example in which inscribed polygons P_1 and P_2 are used for the approximation of O_1 and O_2, respectively. In this case, the Minkowski sum $P_1 \oplus P_2$ is also a polygonal object. Figures 34(d) and 35(d) show the elimination procedure based on Rules 1–2, and a simpler Minkowski sum $P_1 \oplus P_2$. Note that the remaining convolution curves in Figures 34(d) and 35(d) have relatively few redundancies compared with other elimination procedures based on Rules 1–3.

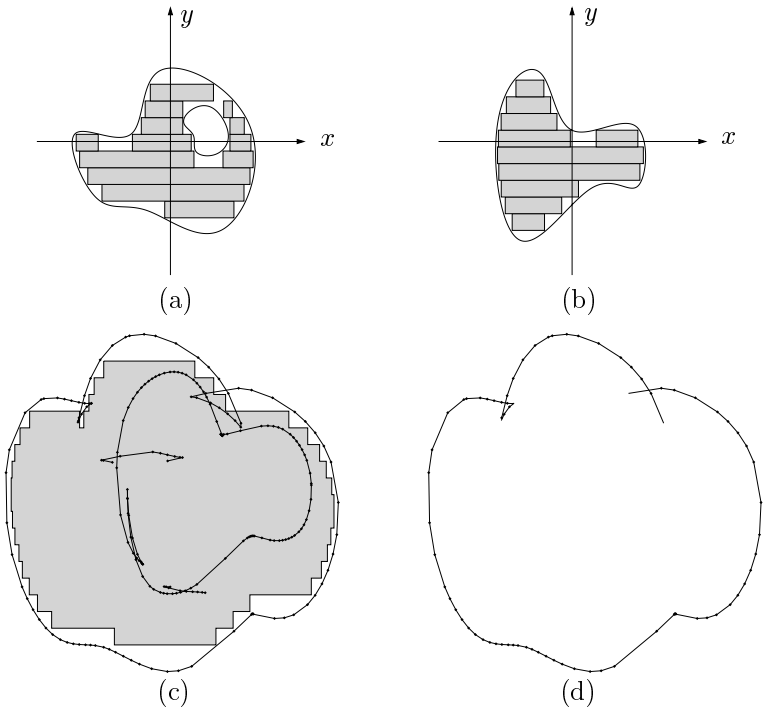


Figure 34: Trimming by a rectilinear subset of the Minkowski sum

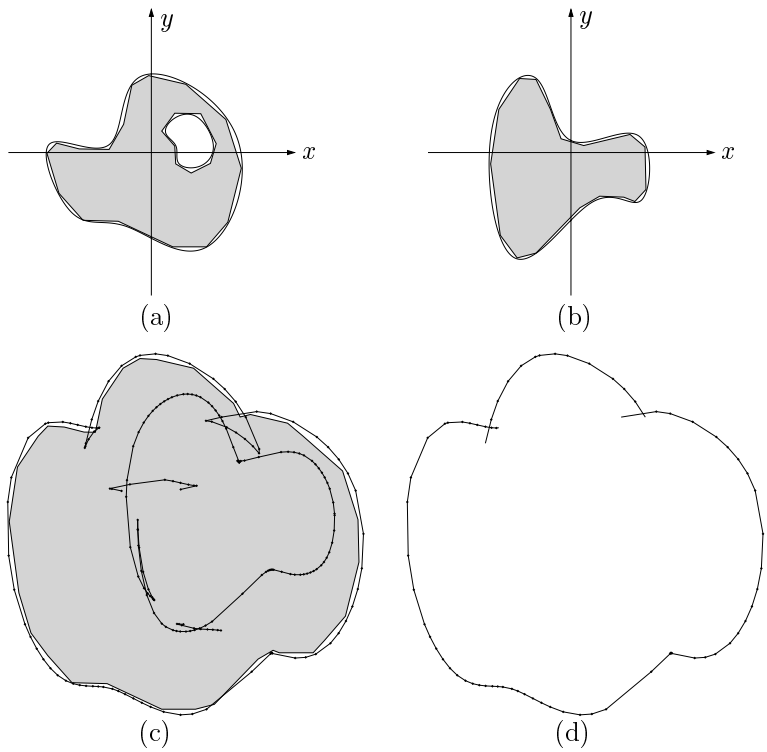


Figure 35: Trimming by a polygonal subset of the Minkowski sum

In Figure 36, we show a sequence of shape transformations from a bird to a butterfly. The intermediate shapes are the Minkowski sums of the bird and butterfly shapes while scaling the bird from 100 % to 0 % and the butterfly from 0 % to 100 %, simultaneously (see also Kaul and Rossignac [24]). Bird and butterfly objects are bounded by piecewise cubic B-spline curves. In this example, we use the QAC method for the curve–curve convolution computation. Thus, the Minkowski sum boundary is a

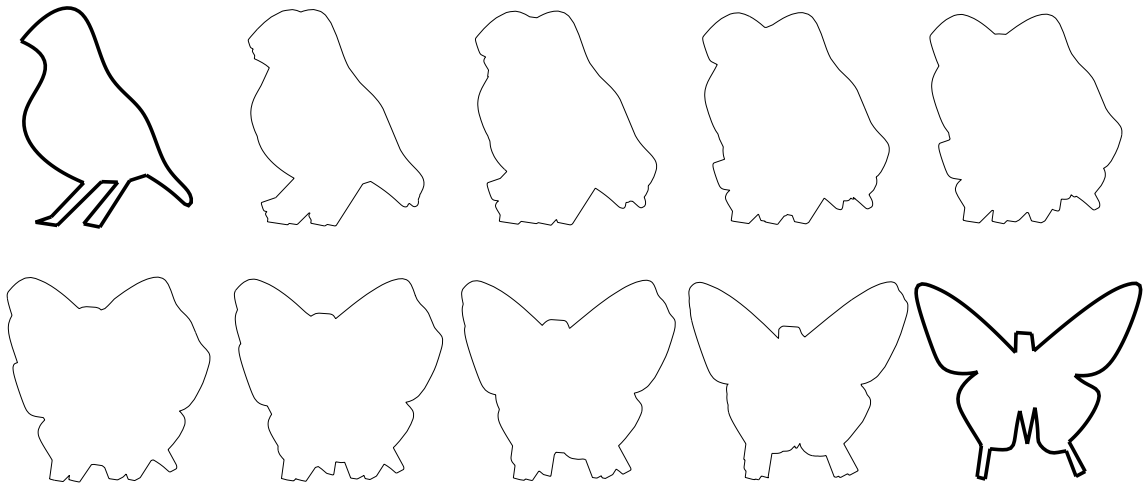


Figure 36: Transformation from bird to butterfly

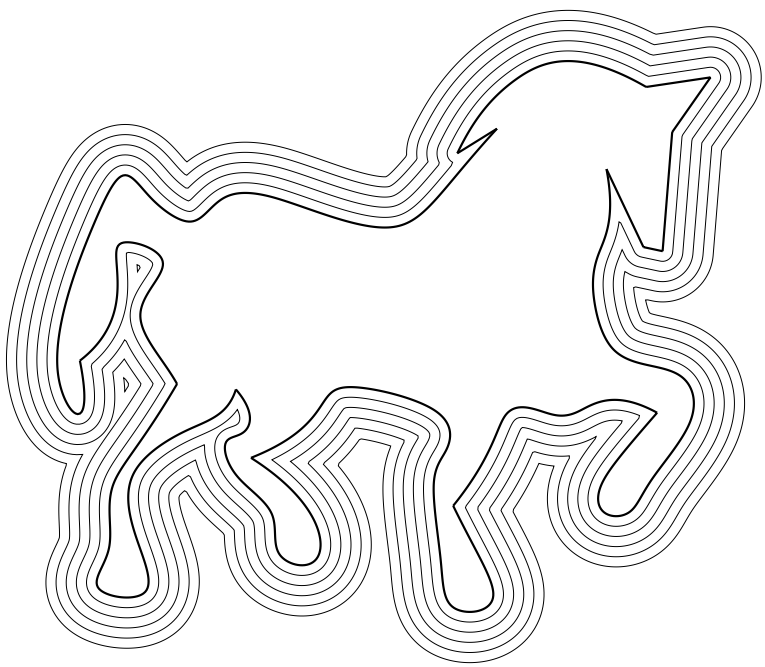


Figure 37: Offsetting

piecewise rational B-spline curve of degree seven. Figure 37 shows the offset boundary curves that are generated by computing the Minkowski sums of the horse-shaped object and the circles with different radii. The horse shape is represented by eight cubic B-spline curves and six line segments, and the circles are represented by rational quadratic curves. Figure 38 demonstrates the generation of C-space obstacles for a robot (Figure 38(b)) and the obstacles consisting of "CSPACE" character shapes (Figure 38(a)). Figures 38(d) and 38(e) show the untrimmed convolution curves and the Minkowski sum boundary, respectively. In Figure 38(f), we verify the computed C-space obstacle by sweeping the robot, while its local reference point follows along the C-space obstacle boundary.

The polygonal approximation may miss some valid loops in the exact boundary of a Minkowski sum, or include some invalid loops. A simple way to resolve this problem might be to approximate the convolution curve with line segments using high precision. However, this approach generates many tiny line segments which cause problems in robustness as well as in computational efficiency. The missing valid loops are due to the tangential and/or multiple intersections of the convolution

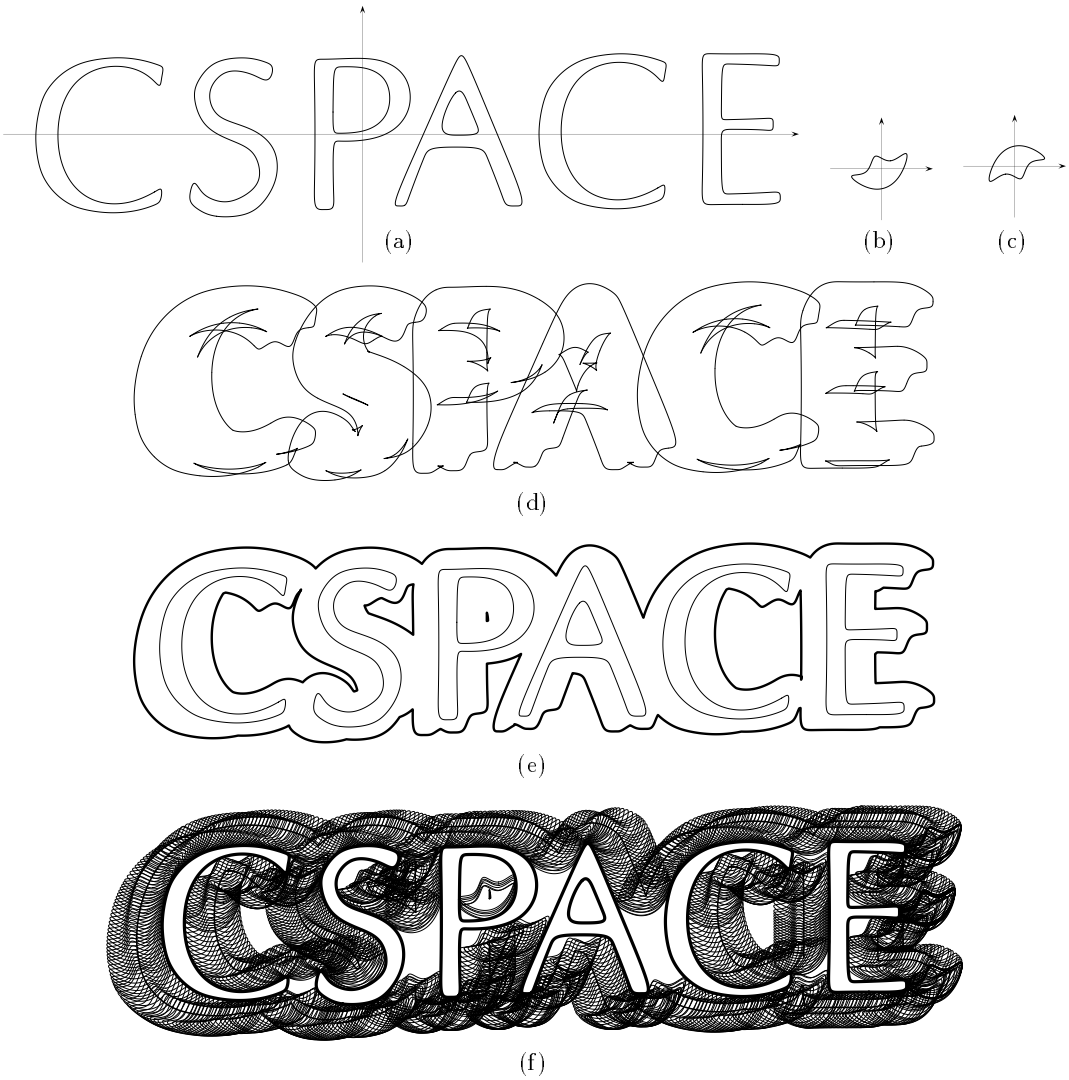


Figure 38: C-space obstacles

curve segments. These degeneracies correspond to the topological changes of the Minkowski sum boundary as the shapes of input objects are slightly changed. For each degeneracy, the corresponding convolution curve segments must be intersected with higher precision to determine a correct topology. However, it is not always possible to determine the correct topological arrangement while making it consistent with the numerical curve intersection data, especially when the curve segments have (almost) tangential/multiple intersections [22].

7 Conclusion

This paper presented new methods to approximate convolution curves. We demonstrated that many techniques developed for offset curve computation can be extended to convolution curve computation. As a result, we suggested several new methods that approximate the convolution curves with polynomial/rational curves, motivated by applications in conventional CAD systems.

In particular, we proposed the techniques based on the curve reparametrization: for example, *quadratic curve approximation* and *tangent field matching*. Quadratic curve approximation and reparametrization based approaches have many advantages such as simple error analysis and output data reduction. We expect that the concepts of quadratic curve approximation and tangent field matching can also be used for many other geometric operations which are closely related to the normal and/or tangent

directions of planar curves.

The 3D extension of convolution curve approximation techniques remains an important problem for future research. Bajaj and Kim [2, 4] showed that 3D offset and convolution of algebraic surfaces are also algebraic; however, their algebraic degrees are very high. Therefore, for the applications in practice, it is very important to approximate the 3D convolution surfaces by polynomial/rational surfaces. Considerable research effort in CAGD is required for the 3D extension of the work presented in this paper.

Qualitative and quantitative comparisons are also made among many convolution curve approximation methods. We believe that these comparison results provide an important guideline for future research in convolution curve computation.

We also demonstrated the effectiveness of the piecewise linear approximation and the plane sweep algorithm in the elimination of redundant parts. Nevertheless, a robust implementation of the plane sweep algorithm for planar curve segments still remains an important open problem.

References

[1] J.-W. Ahn, M.-S. Kim, and S.-B. Lim. Approximate general sweep boundary of a 2D curved object. *CVGIP: Graphical Models and Image Processing*, 55(2):98–128, March 1993.

[2] C. Bajaj and M.-S. Kim. Generation of configuration space obstacles : The case of a moving sphere. *IEEE J. of Robotics and Automation*, 4(1):94–99, 1988.

[3] C. Bajaj and M.-S. Kim. Generation of configuration space obstacles : The case of moving algebraic curves. *Algorithmica*, 4(2):157–172, 1989.

[4] C. Bajaj and M.-S. Kim. Generation of configuration space obstacles : The case of moving algebraic surfaces. *The Int'l J. of Robotics Research*, 9(1):92–112, 1990.

[5] B. Cobb. *Design of Sculptured Surface Using The B–spline Representation.* Ph.D. thesis, University of Utah, Computer Science Department, 1984.

[6] S. Cohen, G. Elber, and R. Bar-Yehuda. Matching of freeform curves. *Computer-Aided Design*, 29(5):369–378, 1997.

[7] G. Elber and E. Cohen. Error bounded variable distance offset operator for free form curves and surfaces. *Int' J. of Computational Geometry and Applications*, 1(1):67–78, 1991.

[8] G. Elber. *Free Form Surface Analysis Using A Hybrid of Symbolic and Numerical Computation.* Ph.D. thesis, Department of Computer Science, The University of Utah, 1992.

[9] G. Elber. *IRIT Version 7.0 Programmer's Manual*, 1997.

[10] G. Elber, I.-K. Lee, and M.-S. Kim. Comparing offset curve approximation methods. *IEEE Computer Graphics & Applications*, 17(3):62–71, May–June 1997.

[11] G. Farin. *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Guide.* Academic Press, San Diego, 4th edition, 1997.

[12] G. Farin. *NURB Curves and Surfaces: from Projective Geometry to Practical Use.* A.K. Peters, Wellesley, Massachusetts, 1995.

[13] R. T. Farouki and V. T. Rajan. Algorithms for polynomials in Bernstein form. *Computer Aided Geometric Design*, 5(3):1–26, 1988.

[14] R. T. Farouki and C. A. Neff. Algebraic properties of plane offset curves. *Computer Aided Geometric Design*, 7(1–4):101–127, 1990.

[15] P. Ghosh and S. P. Mudur. The brush-trajectory approach to figure specification: Some algebraic-solutions. *ACM Trans. on Graphics*, 3(2):110–134, April 1984.

[16] P. Ghosh. A mathematical model for shape description using Minkowski operators. *Computer Vision, Graphics, and Image Processing*, 44(3):239–269, 1988.

[17] P. Ghosh. An algebra of polygons through the notion of negative shapes. *Computer Vision, Graphics, and Image Processing: Image Understanding*, 54(1):119–144, 1991.

[18] P. Ghosh. A unified computational framework for Minkowski operations. *Computers and Graphics*, 17(4):357–378, 1993.

[19] L. Guibas, L. Ramshaw, and J. Stolfi. A kinetic framework for computer geometry. In *Proc. of 24th Annual Symp. on Foundations of Computer Science*, pages 100–111, 1983.

[20] L. Guibas and D. Marimont. Rounding arrangements dynamically. In *Proc. of 11th Annual Symp. on Computational Geometry*, pages 190–199, 1995.

[21] Hansen, A., and Arbab, F., "An Algorithm for Generating NC Tool Paths for Arbitrarily Shaped Pockets with Islands," *ACM Trans. on Graphics*, 11(2):152–182, 1992.

[22] C. Hoffmann. *Geometric and Solid Modeling.* Morgan Kaufmann, San Mateo, 1989.

[23] J. Hoschek and D. Lasser. *Fundamentals of Computer Aided Geometric Design.* A.K.Peters, Wellesley, Massachusetts, 1993.

[24] A. Kaul and J. R. Rossignac. Solid interpolating deformations: Construction and animation of pip. *Computers and Graphics*, 16(1):107–115, 1992.

[25] A. Kaul and R. Farouki. Computing Minkowski sums of plane curves. *Int' J. of Computational Geometry & Applications*, 5(4):413–432, 1995.

[26] K. Kim and G. Elber. New approaches to freeform surface fillets. *The J. of Visualization and Computer Animation*, 8(2):69–80, 1997.

[27] M. Kohler and M. Spreng. Fast computation of the C-space of convex 2D algebraic objects. *The Int' J. of Robotics Research*, 14(6):590–608, 1995.

[28] M. Kosters. Curvature-dependent parametrization of curves and surfaces. *Computer-Aided Design*, 23(8):569–578, 1991.

[29] I.-K. Lee and M.-S. Kim. Primitive geometric operations on planar algebraic curves with Gaussian approximation. *Visual Computing*, T.L. Kunii (Ed.), Springer-Verlag, Tokyo, pages 449–468, 1992.

[30] I.-K. Lee, M.-S. Kim, and G. Elber. Planar curve offset based on circle approximation. *Computer-Aided Design*, 28(8):617–630, August 1996.

[31] I.-K. Lee, M.-S. Kim, and G. Elber. New approximation methods of planar offset and convolution curves. *Geometric Modeling: Theory and Practice*, W. Strasser, R. Klein, and R. Rau (Eds.), Springer-Verlag, Heidelberg, pages 83–101, 1997.

[32] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision free paths among polyhedral obstacles. *Comm. of the ACM*, 22(10):560–570, 1979.

[33] T. Lozano-Pérez. Spatial planning : A configuration space approach. *IEEE Trans. on Computers*, 32(2):108–120, 1983.

[34] F. P. Preparata and M. I. Shamos. *Computation Geometry: An Introduction*, Springer-Verlag, New York, 1985.