Untrimming: Precise conversion of trimmed-surfaces to tensor-product surfaces

Fady Massarwi, Boris van Sosin, Gershon Elber

Department of Computer Science, Technion, Israel

Abstract

Trimmed B-spline surfaces are common in the geometric computer aided design (CAD) community due to their capability to represent complex shapes that can not be modeled with ease using tensor product B-spline and NURBs surfaces. However, in many cases, handling trimmed-surfaces is far more complex than tensor-product (non-trimmed) surfaces. Many algorithms that operate on tensor-product surfaces, such as algorithms toward rendering, analysis and manufacturing, need to be specially adapted to consider the trimming domains. Frequently, these special adaptations result in lack of accuracy and elevated complexity. In this paper, we present an algorithm for converting general trimmed surfaces into a set of tensor-product (typically B-spline) surfaces. We focus on two algorithms to divide the parametric space of the trimmed surface into four-sided quadrilaterals with freeform curved boundaries, which is the first step of the algorithm. Then, the quadrilaterals are parameterized as planar parametric patches, only to be lifted to the Euclidean space using a surface-surface composition, resulting in tensor product surfaces that precisely tile the input trimmed surface in Euclidean space. The algorithm is robust and precise. We show that we can handle complex, industrial level, objects, with numerous high orders and rational surfaces and trimming curves. Finally, the algorithm provides user control on some properties of the generated tensor-product surfaces.

Keywords: Composition, Line-sweep quadrangulation, Optimal quadrangulation, Precise integration, Precise bounding box.

1. Introduction

Tensor product (Bézier, B-spline and NURBs) surfaces are widely used in geometric computer aided design (CAD) due to their simple structure, mathematical form, and powerful geometrical properties that make them intuitive to use. However, they are limited to the rectangular topology, making it difficult to create general 3D objects. That is, the rectangular topology doesn't allow to represent with ease general boundaries, including holes. Due to these limitations of the tensor product surface representation, trimmed-surfaces were introduced [1]:

Definition 1.1. *A* trimmed B-spline surface*, S_t, is a tensor-product B-spline surface, S, whose domain is bounded by a set of trimming B-spline closed curves, C_t. Typically, one, outer boundary trimming curve exists, and other internal trimming curves define holes in the parametric domain. The orientation of the trimming curves is defined such that the trimmed-surface lies on the same side (e.g right) of the trimming curves, as we move along the trimming curve.*

A common method for creating CAD models is by applying Boolean set operations between simpler models [2, 3, 4], where the intersection curves between the surfaces of the models define the trimming curves. In the ensuing discussion and unless otherwise stated, we will refer to a trimmed B-spline surface/curve while it can also be a Bézier or a NURBs surface/curve.

Compared to tensor-product surfaces, trimmed-surfaces ease the process of representing results of Boolean set operations, and allow simpler modeling of complex shapes. However, there are difficulties in using trimmed-surfaces compared to tensor product surfaces. Due to the complex parametric boundaries and the non-rectangular topology, powerful geometrical properties of the B-spline representation, such as the convex hull property [1], are less faithful to trimmed-surfaces than to tensor-product surfaces. Algorithms designed for tensor product B-spline surfaces, such as algorithms toward rendering, manufacturing and analysis, do not directly extend to trimmed-surfaces and require special treatments, if at all feasible.

A recent development in physical analysis, Iso Geometric Analysis (IGA) [5], performs the analysis directly in spline spaces over the spline surfaces of the models, which practically means models with trimmed-surfaces. IGA requires precise integration over the surfaces, among others. However, integration over trimmed B-spline basis functions is a challenging non-trivial task, in the general case. Approximating trimmed-surfaces by piecewise-linear elements, in order to simplify the integration process, will result in loss of accuracy and might affect the quality and convergence of the analysis. Methods to precisely integrate over the trimming domains are required, in order to have a complete and accurate IGA over trimmed-surfaces. One way to achieve this goal, is by first converting the trimmed-surfaces to tensor-products. In this work, we present the untrimming process not only as a geometry conversion process but also as an intermediate representation to precisely integrate over trimmed domains and hence is a precise fit to the IGA approach, for trimmed surfaces.

In this paper, we introduce an algorithm with two variations for converting general trimmed-surfaces into a set of tensor product B-spline surfaces, a process we denote as *untrim-*

ming. The algorithm is robust and precise[1], and is able to handle complex industrial models composed of thousands of trimmed-surfaces. The algorithm first divides the trimmed parametric domain into quadrilaterals with freeform boundary curves while precisely preserving the trimming curves. Then, the quadrilaterals are parameterized into planar patches. Finally, these (tensor-product planar) patches are lifted to the Euclidean space via a symbolic surface-surface composition [6, 7, 8]. The end result is a precise tiling of the original trimmed surface, by tensor product surfaces, albeit of higher degrees. The main contribution of this work includes two variations of a quadrangulation algorithm of freeform (trimmed) domain. The first variation of the algorithm builds the quadrangulation using a fast line-sweep based algorithm, and the second variation builds the quadrangulation based on the minimization of a given weight function, which enables some control over different desired properties of the generated output. The untrimming algorithm can handle rational and arbitrary order trimming curves and trimmed-surfaces.

We like to emphasize that the presented conversion is precise for each individual trimmed surface. If cracks (black holes) exist (i.e. due to imprecise Boolean Set operations) between different trimmed surfaces, these cracks will be precisely reconstructed, as this work focus on the precise reconstruction of individual trimmed surfaces as tensor products.

The rest of this document is organized as follows. Section 2 discusses related work and in Section 3, we describe our untrimming algorithm, with its two variations. In Section 4, we present some results of untrimming of several trimmed surfaces' domains and CAD models, and compare the different proposed methods. Finally, in Section 5, we conclude and discuss future planned research.

2. Related Work

Several studies have proposed algorithms for generating quad meshes, such as [9, 10, 11, 12]. However, these algorithms have been developed for triangular surface meshes, and are not easily adapted to trimmed B-spline surfaces with high-order B-spline trimming curves. A method for converting trimmed NURBs surfaces to Catmull–Clark subdivision surfaces is described in [13]. The method in [13] is limited to bi-cubic NURBs.

Other studies focused on rendering of trimmed-surfaces. Schollmeyer et al. [14] proposed a fast and direct method for rendering trimmed-surfaces that is aimed to avoid the inaccuracies introduced if the trimming curves are not precise. Martin et al. [15] proposed a ray tracing algorithm for trimmed-NURBs and provided an algorithm for ray-NURBs intersection that is based on hierarchical pruning and numerical refinements. Both methods, [14] and [15], exploit algorithms for a point inclusion test in the trimmed parametric domain. However, these methods allow a pixel error approximation in the trimming curve point inclusion test, and thus appropriate for rendering only. Further, it is unclear how can these methods be extended to precisely handle trimmed surfaces, for general, non-rendering, tasks.

Approximating trimmed-surfaces by a set of primitives have been studied, for example, in [16, 17, 18], where the challenge is to minimize the number of approximating triangles with respect to a user defined error tolerance. A common problem when tessellating trimmed surfaces, is the generation of cracks and gaps along common trimming boundaries between neighboring trimmed-surfaces (also known as "black holes"). Several studies have addressed the cracks problem [19, 20] and suggested methods for fixing the tessellation errors and stitching the cracks. The cracks' problem could have potentially been avoided if the trimming of the trimmed surface has been precise and the surface is precisely converted to a set of tensor product surfaces. Unfortunately, the computation of the surface-surface intersection curves, as part of Boolean set operations, are rarely within machine precision.

The conversion of trimmed-surfaces into tensor-product surfaces have been studied in [6, 21, 22, 23, 24]. [22] uses curvature oriented segmentation in order to obtain bi-cubic Bézier patches, but [22] can't handle rational trimmed surfaces, and approximate them by a bi-cubic or bi-quintic polynomial surfaces. Further, the mapping process of the resulted quadrilaterals from the parametric domain to the Euclidean space is not precise and utilizes interpolation methods. [24] partitions the parametric space into quadrilaterals using feature points of the trimming curves, but it is designed to be precise only for bi-cubic polynomial B-spline surfaces, in an effort to reduce the degrees of the outcome. The trimming domain is partitioned in *turn* points, locations on the trimming curve $C_t(t) = (u(t), v(t))$ that satisfies $|u'(t)| = |v'(t)|$, and results in over-partitioning of the domain. Further, a closed piecewise C^1 discontinuous trimming curve may have no location for which $|u'(t)| = |v'(t)|$, cases that are not discussed in [24] while they are handled in this work. Hamann et al. [21] employs a scan-line based algorithm for partitioning the parametric space to a rectangular domains. However, their method involves triangulation and Voronoi-diagram computation over the trimmed-parametric domain, which makes it less robust and complex to implement. Also, [21] assumes that the ruling between any two monotone regular, (non vanishing derivative), curves always produces a regular (consistent Jacobian) surface, which we show to not necessarily be the case (and also show a remedy). Hui et al. [23] also employs a Voronoi-diagram approach for partitioning the trimmed domain into simpler cells, and improves the method proposed in [21] by using feature point matching approach rather than a scan-line approach in order to reconstruct four-sided surfaces. However, [23] doesn't provide methods for mapping the resulted tensor-product surfaces from the parametric space to the Euclidean space. Finally, while [22, 23, 24] recognize the importance of only regular patches in the output, they do not discuss how to achieve this goal.

In [6] several applications of functional composition of B-spline curves and surfaces have been introduced. Following [6], we use a symbolic surface-surface composition as the final step to lift the generated quadrilaterals from the parametric space to the Euclidean space. [6] discusses an algorithm for converting a trimmed-surface to tensor-product surfaces. However, the algorithm in [6] does not offer a general quadrangulation and hence is limited to simple topologies and can't handle industrial level trimmed surfaces.

[1] In this work, precise denotes a precision that approaches the accuracy of the hardware (machine precision).

None of the above methods is capable of precisely handling general models with high order rational trimming curves and trimmed-surfaces. Also, previous methods, with the exception of [6], don't involve general symbolic computation to provide precise and robust partition of the parametric domain, and a precise composition of patches in the parametric domain with the surface, into the Euclidean space. Finally, these previous methods provide no user control on the quality and different properties of interest over the result; knowing that there could be several mappings of the trimmed domain into quadrilateral sub-domain, it might be desired to provide some user control on the algorithm's output. Interested in precise and efficient integration, in this work, we focus on ensuring positive Jacobian in the interior of the resulted quadrilaterals, and care less about "nice" quadrilaterals. Though "nice" quadrilaterals can be achieved by using the second variation of the proposed quadrangulation algorithm.

3. The Untrimming algorithms

Having a trimmed surface, S_t, defined over a parametric domain, D, of tensor product surface, S, with a set of trimming curves, C_t, the general steps of the untrimming algorithm are described in Algorithm 1. Step 1 in Algorithm 1 consists

Algorithm 1 : The untrimming algorithm

Input:
S_t, a trimmed (B-spline) surface defined over the parametric domain, D, of a tensor product surface, S, and a set of (B-spline) trimming curves, C_t, in D;

Output:
$\mathcal{S}$, a set of tensor product (B-spline) surfaces precisely spanning S_t, in the Euclidean space;

Algorithm:

1: $\mathcal{S}_t^{split}$:= divide S_t into smaller simple trimmed Bézier surfaces, each having no holes;
2: $\mathcal{S} := \emptyset$;
3: **for all** $S_t^i \in \mathcal{S}_t^{split}$ **do**
4: Q_i^p := Freeform planar quadrilaterals tiling domain D^i of S_t^i, consisting of one trimming curve;
5: Q_i^m := Merge adjacent quadrilateral patches in Q_i^p whenever possible;
6: $\mathcal{S}_i := Q_i^m$ lifted into the Euclidean space, via surface-surface compositions: $S_t^i(Q_j), \forall Q_j \in Q_i^m$;
7: $\mathcal{S} := \mathcal{S} \cup \mathcal{S}_i$;
8: **end for**
9: return $\mathcal{S}$;

of splitting the input trimmed surface, S_t, into simple trimmed Bézier surfaces, $\mathcal{S}_t^{split}$, having only a single (outer) trimming curve. Splitting the input trimmed surface, S_t, into simple trimmed Bézier surfaces is achieved in two steps: First, the trimmed B-spline surface is divided at all its internal knots to yield a set of trimmed Bézier surfaces. Due to surface-surface composition limitations [6, 7, 8], applied in step 6, Q_j in $S_t^i(Q_j)$ cannot cross knot lines of S_t^i (or otherwise Q_j must be split along the knot lines of S_t^i and re-quadrangulated). We ensure no such crossings occur by forcing S_t^i to be a Bézier surface. Then, for each hole, $h \in C_t$, we select a representative domain point, p, that lies inside h, and further divide S_t along the u (or v) direction at the u (v) value of p. See Figure 1 for an example.

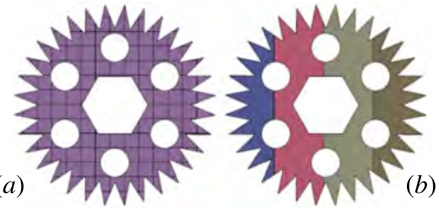

Figure 1: A teeth wheel (from Figure 14) which is a single Bézier surface with a complex trimming curve (a). The surface is divided at all interior holes of the trimming curves, yielding (b).

In step 4, each trimming curve of each surface $\mathcal{S}_t^{split}$ is converted into a set of quadrilaterals in the parametric domain. We propose two variations for converting the domain of a trimmed Bézier surface, S_t^i, with a single trimming loop, into a set of freeform shaped quadrilaterals, Q_i^p. These two quadrangulation variations differ only in step 4 and these quadrangulation algorithms are described in Section 3.1. In step 5, adjacent planar topologically-rectangular surfaces are merged as much as possible. More details on this merge process are described in Section 3.2. All these generated quadrilaterals are in their respective D^i's, the domain of S_t^i, and in step 6, these planar patches are lifted into the Euclidean space, via surface-surface composition operations; the composition algorithm is briefly discussed in Section 3.3.

3.1. Domain division into Quadrilaterals

Consider a trimmed Bézier surface, $S_t(u, v)$, and let $C_t(t)$ be the sole trimming loop curve in its trimming domain, D. $C_t(t)$ is assumed to be simple, i.e. no self-intersections, and is also regular, or $\|C_t'(t)\| > 0$. We propose two algorithms for precisely mapping the domain spanned by $C_t(t)$ to a set of mutually exclusive quadrilaterals in D, that can have freeform boundaries. $C_t(t)$ is typically a B-spline curve and we assume the introduction of no new interior (Steiner) points, in the quadrangulation process. The domain of each quadrilateral is then parameterized as a tensor product planar B-spline surface, using the four curves bounding the quadrilateral with the aid of Boolean Sum [1]. Finally, we also portray conditions under which the Boolean Sum succeeds, i.e. the resulting patches are regular. We now discuss both algorithms, in Sections 3.1.1 and 3.1.2, respectively.

3.1.1. Line-sweep quadrangulation algorithm

Once the sole trimming loop curve, $C_t(t)$, has been extracted from the trimmed surface, the simple closed loop curve needs to be converted to a set of mutually exclusive freeform quadrilaterals. In this section, we use a line-sweep approach, which is used in various algorithms for partitioning planar shapes, such as polygon triangulation [25]. Intuitively, the line-sweep algorithm works by sweeping the curve with a vertical (sweeping) line moving from left to right, and finding all the points on the curve at which the sweeping line encounters one of the following events: a start of a new patch on the inside of the curve, the end of an existing patch, a patch being split into two, or two patches merging into one. Examples for the start, merge, end and split events are shown in Figure 2. Additionally, a practical implementation of the line-sweep algorithm should handle line-sweep events such as vertical lines, cusps, and C^1 discontinuities which cause extrema with respect to the sweeping line. The events discovered by the sweep are then matched to form pairs of curve segments of the original curve, which will be referred to as slices:

Definition 3.1. *A* slice *is a pair of curve segments* $C_t^1(p) = (x_1(p), y_1(p)),\ p \in [p_s,\ p_e]$, $C_t^2(r) = (x_2(r), y_2(r)),\ r \in [r_s,\ r_e]$ *that have the following properties:*

1. C_t^1 *is a curve segment of* $C_t(t)$.
2. C_t^2 *is a curve segment of* $Reverse(C_t(t))$, *where* $Reverse(C(t))$ *is the reversed parametrization of* $C(t)$.
3. $x_1(p_s) = x_2(r_s)$.
4. $x_1(p_e) = x_2(r_e)$.

In other words, a slice consists of a pair of curves, one above the other, sharing the same x-span. These pairs of curve segments form the boundaries of ruled surfaces which are the output of the algorithm. The algorithm is formally defined in Algorithm 2.

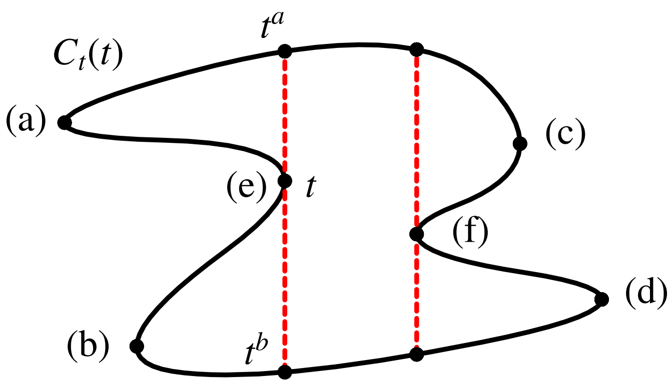

Figure 2: Start (a,b), End (c,d), Merge (e) and Split (f) events on a closed curve $C_t(t)$. Also, the (e) merge event is shown (at parameter t), with the other slice start/end points the line-sweep generated from the t event (denoted by t^a for above and t^b for below).

In steps 3, 10, 12, 18, 19, Algorithm 2 needs to find the zero set of a scalar curve. Our implementation of the line-sweep algorithm uses an efficient method of solving the zero-set of a univariate as described in [26]. In steps 7, 13, 14, 16, 20, 21 of Algorithm 2, we create ordered pairs of parameter values which define the limits of the top or bottom curve segments of the slices. Note that in start (and end) events, the top and bottom curve segments start (and end, resp.) locations are the same. In step 28, the output ruled surfaces are formed. Each slice consists of four curves, two of which are (vertical) lines, from $C_t(p_s)$ to $C_t(r_s)$, and from $C_t(p_e)$ to $C_t(r_e)$. A Boolean sum constructor [1] adds the ruling between these two segments of C_t to the ruling between the two vertical lines, only to subtract a bilinear between the four corner locations. Herein however, the Boolean sum of these four curves degenerates into a ruled surface as the ruling between the vertical lines is equal to the Bilinear and cancels it.

Algorithm 2 : line-sweep based algorithm

Input:
$C_t(t) = (x(t), y(t)),\ t \in [0, 1]$, a closed planar curve assumed to be oriented clockwise;

Output:
Q, a set of planar freeform quadrilaterals covering the interior of $C_t(t)$;

Algorithm:

```
1:  S_s:=∅; // Initialize sets of slice starts,
2:  S_e:=∅; // and slice ends.
3:  E:={t ∈ [0,1] | x'(t) = 0};
4:  for all t ∈ E do
5:    switch (ClassifyEvent(C_t, t) // Algorithm 3)
6:      case Start:
7:        S_s:=S_s ∪ {(t, t)};
8:      case Merge:
9:        // Location on C_t directly above C_t(t):
10:       t^a:=argmin_{r∈[0,1]}(y(r) | x(r) = x(t) && y(r) > y(t));
11:       // Location on C_t directly below C_t(t):
12:       t^b:=argmax_{p∈[0,1]} (y(p) | x(p) = x(t) &&
              y(p) < y(t));
13:       S_s:=S_s ∪ {(t^a, t^b)};
14:       S_e:=S_e ∪ {(t^a, t), (t, t^b)};
15:     case End:
16:       S_e:=S_e ∪ (t, t);
17:     case Split:
18:       t^a:=argmin_{r∈[0,1]}(y(r) | x(r) = x(t) && y(r) > y(t));
19:       t^b:=argmax_{p∈[0,1]} (y(p) | x(p) = x(t) &&
              y(p) < y(t));
20:       S_s:=S_s ∪ {(t^a, t), (t, t^b)};
21:       S_e:=S_e ∪ {(t^a, t^b)};
22:   end switch
23: end for
24: Q:=∅; // Initialize set of output quadrilaterals.
25: for all (t_s^a, t_s^b) ∈ S_s do
26:   // Find pair (t_e^a, t_e^b) such that t_e^a follows t_s^a, in the parametric domain.
27:   (t_e^a, t_e^b):=argmin_{(t^a,t^b)∈S_e}(t^a | t^a > t_s^a);
28:   R:=RuledSurface(C_t(t), t ∈ [t_s^a, t_e^a],
          ReverseCurve(C_t(t)), t ∈ [t_s^b, t_e^b]);
29:   Q:=Q ∪ {R};
30: end for
31: return Q;
```

Algorithm 2 can be implemented with a computational complexity of $O(n \log(n))$ (n being the number of line-sweep events), as the line-sweep events can be computed all at once (by finding the solutions to $x'(t) = 0$) and then sorted in x. Later, in step 27, the limits of each slice can also be computed by sorting the set $\mathcal{S}_s$. Then, $O(n)$ operations on a sorted list of $O(n)$ points can be done in $O(n \log(n))$ time. Algorithm 3 presents the logic behind the events' classification.

We now discuss the properties of the patches that the line-sweep algorithm produces, and present a way of guaranteeing that the Jacobian of the patches does not change its sign in the interior of the patch. For the sake of the discussion, let

Algorithm 3 : ClassifyEvent

Input:

$C_t(t) = (x(t), y(t)),\ t \in [0, 1]$), a closed planar curve assumed to be oriented clockwise;

$t \in [0, 1],\ x'(t) = 0$;

Output:

A decision whether the point $C_t(t)$ is a start, end, merge, or split event;

Algorithm:

```
if y'(t) > 0 then
    if x'(t − ε) < 0 && x'(t + ε) > 0 then
        return Start;
    else if x'(t − ε) > 0 && x'(t − ε) < 0 then
        return Merge
    end if
else if y'(t) < 0 then
    if x'(t − ε) > 0 && x'(t − ε) < 0 then
        return End;
    else if x'(t − ε) < 0 && x'(t − ε) > 0 then
        return Split;
    end if
else
    return Error // Both x'(t) and y'(t) are zero
end if
```

us assume that the pair of curves, $C_t^1(t)$ and $C_t^2(t)$, in the slice produced by the line-sweep algorithm undergo an affine transformation of their parametric domains so that the domains of both curves are $[0, 1]$. According to the definition of the algorithm, the pairs of curve segments which define the resulting patches (ruled surfaces) cannot contain extreme points with respect to the sweeping line (except possibly at the endpoints of the curves, where the Jacobian can indeed vanish), because Algorithm 2 subdivides $C_t(t)$ at such points.

We start by noticing that there exist a regular reparametrization $\tau(t)$ of the ruling $R(t, v) = vC_t^1(t) + (1 - v)C_t^2(\tau(t))$ between a pair of curves, $C_t^1(t) \subset C_t(t)$ and $C_t^2(t) \subset C_t(t)$, of a slice, for which the Jacobian does not change its sign in the interior. A vertical parametrization (i.e. $x_1(t) = x_2(\tau(t))$) ensures that $R(t, v)$ is indeed regular in its interior, simply because $\frac{\partial R}{\partial v}$ is a non zero vertical vector ($C_t(t)$ is self-intersection free), and $\frac{\partial R}{\partial t}$ is a convex blend of two vectors, for which both $x_1' > 0$ and $x_2' > 0$, as these two curves are regular (as $C_t(t)$ is regular) and x-monotone in their interior.

The problem of reparameterizing a pair of curves in such a way that the ruled surface between them is regular, or the Jacobian does not change its sign, has been addressed, for example, in [27]. However, such algorithms tend to be computationally expensive and they raise the degree of the outcome due to the explicit composition operation, $C_t^2(\tau(t))$ (unless $\tau(t)$ is piecewise linear, in which case continuity is affected).

Because we are already expecting to raise degrees due to the surface-surface composition operations (recall step 6 in Algorithm 1), and seeking a simpler and more efficient solution, we simply form a ruled surface from the pair of curves of each slice (as the slices are returned from the line-sweep algorithm) and check whether or not their Jacobian changes signs. The determinant of the Jacobian of ruling, $R(t, v)$,

$$|J(t, v)| = \left|\frac{\partial R}{\partial t} \times \frac{\partial R}{\partial v}\right|,$$

is symbolically computed as a scalar bi-variate spline function. Then, by examining the coefficients of $|J|$, or if more precision is sought, by computing the extreme values of $|J|$ via the simultaneous zeros of

$$\frac{\partial |J|}{\partial t} = \frac{\partial |J|}{\partial v} = 0,$$

we can answer whether or not the Jacobian changes sign.

If the Jacobian does change sign, we subdivide $C_t^1(t)$ and $C_t^2(t)$ in a middle x value of the slice by intersecting the curves with a vertical line, and continue recursively on each of the resulting slices. Since the slices are subdivided vertically (i.e. at the same x value), as the slices get narrower, the v direction of the ruled surfaces approaches the vertical parametrization. Given a finite $\varepsilon > 0$, we prove that this recursive subdivision process terminates after a finite number of subdivision steps, and results in patches in which the Jacobian does not change sign in their interior, within an ε neighborhood from the start and end of the slice.

Lemma 3.1. *Consider the pair of regular non-intersecting curves, $C_t^1(p) = (x_1(p), y_1(p))$ and $C_t^2(r) = (x_2(r), y_2(r))$, $p, r \in [0, 1]$, in a slice that results from the line-sweep algorithm. Let $R(t, v)$ be a ruled surface between these curves of the form*

$$R(t, v) = vC_t^1(t) + (1 - v)C_t^2(t), \quad v, t \in [0, 1].$$

Given $\varepsilon > 0$, in the sub-domain $t \in [\varepsilon, 1-\varepsilon]$, there is a minimum width w of vertical subdivisions of C_t^1 and C_t^2 beyond which the determinant of the Jacobian of $R^i(t, v)$ is guaranteed not to change signs.

Proof. Let $R^i(t_i, v)$ is an intermediate ruled surface produced by the vertical subdivision process. The Jacobian of the ruled surface $R^i(t, v)$ is $\frac{\partial R^i}{\partial t_i} \times \frac{\partial R^i}{\partial v} = (vC_{t_i}^{1\prime}(t_i) + (1 - v)C_{t_i}^{2\prime}(t_i)) \times (C_{t_i}^1(t_i) - C_{t_i}^2(t_i))$. $\frac{\partial R^i}{\partial v} = C_{t_i}^1(t_i) - C_{t_i}^2(t_i)$ is an isoparametric line of R and $\frac{\partial R^i}{\partial t_i} = vC_{t_i}^{1\prime}(t_i) + (1 - v)C_{t_i}^{2\prime}(t_i)$ is a convex linear combination of the tangents of the curves $C_{t_i}^1$ and $C_{t_i}^2$.

Neither vectors in the cross product can be zero: $vC_{t_i}^{1\prime}(t_i) + (1 - v)C_{t_i}^{2\prime}(t_i) \neq 0$ because both $C_{t_i}^{1\prime}(t_i)$ and $C_{t_i}^{2\prime}(t_i)$ must be strictly positive in the x direction, and $C_{t_i}^1(t_i) - C_{t_i}^2(t_i) \neq 0$ because the curves do not intersect in $[\varepsilon, 1 - \varepsilon]$; Hence, if the two vectors in the cross product of the Jacobian are not in the same direction, we complete our proof.

According to the properties of the line-sweep algorithm, the curves $C_{t_i}^1$ and $C_{t_i}^2$ are both monotone with respect to the x axis, and can't be vertical in the sub-domain $[\varepsilon, 1-\varepsilon]$ (i.e. $x_1^{i\prime} > 0$ and $x_2^{i\prime} > 0$). Furthermore, the slope of the two vectors $C_{t_i}^{1\prime}(t_i)$ and $C_{t_i}^{2\prime}(t_i)$ has some finite upper bound slope U, for $t \in [\varepsilon, 1 - \varepsilon]$. On the other hand, there is a finite lower bound to the vertical distance between non-intersecting curves $C_{t_i}^1(t_i)$ and $C_{t_i}^2(t_i)$: $\Delta Y_{\min} = \min_t(\{|y_1^i(t_i) - y_2^i(t_i)|\})$, which gives a lower bound of $\frac{\Delta Y_{min}}{w}$ to the slope of a constant t isoline in R^i. Therefore, if we choose a value of w so that $\frac{\Delta Y_{min}}{w} > U$, we guarantee that the determinant of the Jacobian of $R^i(t, v)$ cannot have zero crossings in $[\varepsilon, 1 - \varepsilon]$. □

An example of a patch from the elephant shape used in our tests (see Figure 10) which needed to be subdivided due to improper Jacobian can be seen in Figure 3(a), and the patches that result from the subdivision can be seen in Figures 3(b) and 3(c). Singularities can, however, occur at the corners of the resulting quadrilaterals, where the Jacobian can vanish (but not change signs). These singularities occur due to locations of C_t tangent to the vertical sweep line and hence boundaries of the quadrilaterals.

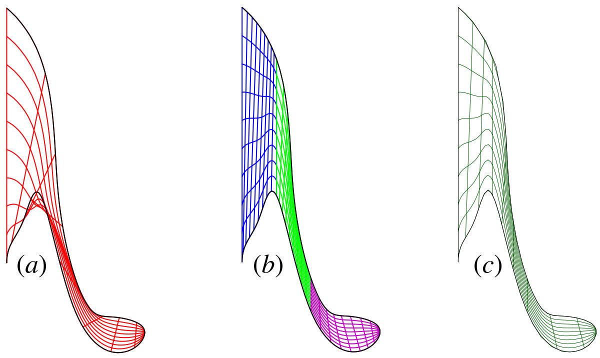

Figure 3: (a) The tail patch of the elephant shape (from Figure 10) produced as a single slice by the line-sweep algorithm, as both the top and bottom boundary curves are monotonous with respect to the x direction. Note that the isoparametric curves of the ruled surface between the two curves of the slice intersect with the boundary curve, indicating that the Jacobian of the patch changes sign. (b) The tail patch of the elephant shape after subdivision. As can be seen from the image, the original patch was subdivided into three smaller patches (i.e. the patch was subdivided into two patches, and one of the new patches was again subdivided in two). Their isoparametric curves do not intersect the boundary curve or each other, indicating that the Jacobian does not change signs. (c) shows the final result after merge.

We would like to point out that the presented line-sweep algorithm is similar to the one proposed in [21], but our preprocessing step of subdividing the trimmed surfaces until they no longer have internal trimmed loops makes it significantly simpler to implement. Additionally, we guarantee that all resulting quadrilaterals are regular, and have a Jacobian that doesn't change signs, without the need for full polynomial reparametrization as in [27] (which also raises the degree of the curves).

3.1.2. Minimal weight function quadrangulation algorithm

Existing methods for converting trimmed surfaces into tensor-product surfaces suffer from having no flexibility nor user control over properties of interest, such as regularity, conformity, and uniformity, over the resulted output surfaces. Having such ability might be desired to control the resulted quadrilaterals. We introduce a quadrangulation algorithm that results in quadrilaterals that best minimizes a user provided weight function. The minimal weight algorithm, unlike virtually all previous related algorithms, including the presented line-sweep algorithm, uses only intrinsic properties of the trimming curves, and is therefore independent of rotation. A minimal weight algorithm for the *triangulation* of a general polygon has been introduced in [28]. We extend the method proposed in [28] and develop a minimal weight *quadrangulation* algorithm for general polygons. One should note that a polygon with odd number of edges cannot be covered using quadrilaterals only. The algorithm will generate quadrilaterals whenever possible and triangles otherwise[2], striving to minimize the number of triangles in the result. We reduce the curve quadrangulation algorithm to a polygon quadrangulation one, and map the result of the polygon quadrangulation algorithm back to the curve's domain. The polygon quadrangulation algorithm scheme is now discussed:

Consider a weight function $W(Q, P)$ that assigns a scalar weight to a quadrilateral Q in a polygon P having n vertices $V_i, i = 1..n$. The algorithm finds $\mathcal{W}(P)$, a weight of a quadrangulation of P that minimizes the sum of all W, for all the Q's that tile P. Suppose we have computed the minimal quadrangulation for all sub polygons of P having less than n vertices. Then, to compute the minimal weighted quadrangulation for P, we do the following: We know that edge (V_1, V_n) will be connected to one or two other vertices forming a triangle or a quadrilateral. Suppose edge (V_1, V_n) is connected to vertices V_i and V_j where $i \leq j$. Then, P is divided into four parts: the quadrilateral $Q = (V_1, V_i, V_j, V_n)$, and three sub polygons, $P_1 = (V_1, ..., V_i), P_2 = (V_i, ...V_j), P_3 = (V_j, ..V_n)$ (P_2 degenerates in case $V_i = V_j$). See Figure 4. $P_i, i = 1..3$ have less than n vertices each. By assumption, we have computed their minimal weighted quadrangulation, $\mathcal{W}(P_i), i = 1..3$ each. Then, the weight for this arrangement is $\mathcal{W}(P_1) + \mathcal{W}(P_2) + \mathcal{W}(P_3) + W(Q, P)$. To compute the globally minimal weight for P, we simply check all possibilities of V_i, V_j, where $i = 2..n-1, j = i, n-1$, (again, note i can be equal to j, in which case a triangle V_1, V_i, V_n is formed). The entire process is described in Algorithm 4. Having a trimmed surface, S_t, with only one trimming

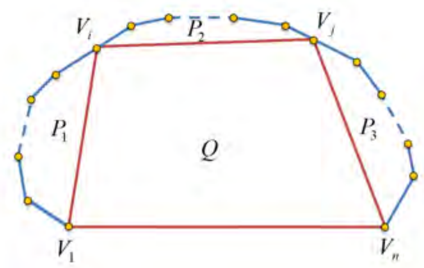

Figure 4: Step of minimal weighted polygon quadrangulation on polygon P. P is divided into four parts: the quadrilateral Q, formed by connecting vertices V_1 and V_n to vertices V_i and V_j, and three sub polygons P_1, P_2, P_3. Assuming the minimal weights for P_1, P_2, P_3 are computed. The weight of this arrangement is the sum of $W(Q, P)$ and the minimal weights of P_1, P_2, P_3. The minimal weight of P is determined by traversing all the possibilities of V_i, V_j.

curve, C_t, a *representative polygon*, P, of C_t is defined as follows (See also Figures 5 and 6):

Definition 3.2. *A* representative polygon*, P, of a closed regular piecewise C^3 continuous parametric curve, C_t, is a polygon with vertices that are sampled from C_t, in a parametric order, as follows:*

1. *Sampling all extreme and inflection points of the curvature κ of C_t, by finding the zeros of the univariate equation $< (\kappa N)', \kappa N >= k' k$ for all C^3 curve segments, where N is the unit normal of C_t. Note $< (\kappa N)', \kappa N >= (\frac{1}{2}(\kappa N)^2)'$ is rational if C_t is.*

[2] Triangles will be converted to singular (at the boundary only) tensor product patches.

Algorithm 4 : PolyQuadrangulate: minimal weight polygon quadrangulation

Input:
P, a representative polygon, having vertices $V_i, i = 1,..n$;
W, quadrilateral weight function;
Output:
$(\mathcal{Q}, w)$, set of quadrilaterals $\mathcal{Q}$ tiling P with minimal weight, w; $\mathcal{W}(\mathcal{Q}) = w$;
Algorithm:

```
if n ≤ 4 then
    w := W(P, P);
    Q := {P};
    return (Q, w);
end if
w := ∞;
for all i ∈ (2..n − 1) do
    for all j ∈ (i..n − 1) do
        P1 := {V1, .., Vi}
        P2 := {Vi, .., Vj}
        P3 := {Vj, .., Vn}
        Qi,j := {V1, Vi, Vj, Vn}
        // Recursive invocations:
        (Q1, w1) := PolyQuadrangulate(P1, W);
        (Q2, w2) := PolyQuadrangulate(P2, W);
        (Q3, w3) := PolyQuadrangulate(P3, W);
        wi,j := w1 + w2 + w3 + W(Qi,j, P);
        if wi,j < w then
            w := wi,j;
            Q := Q1 ∪ Q2 ∪ Q3 ∪ {Qi,j};
        end if
    end for
end for
return (Q, w);
```

2. *Sampling all actual C^1 discontinuities of C_t, by examining the multiplicities of C_t's knots and the control polygon before and after that parameter value.*

Definition 3.3. *Let $[u_s, u_e]$ be the domain of C_t, and let P's vertices be $\mathcal{V}_p = \{V_i\}$, $i = 1..n$ where $V_i = C_t(u_i), u_i \in [u_s, u_e]$, $i = 1..n$. Then, each edge $e_i = (V_i, V_{i+1})$ of P is a representative of the curve segment $C_t^i = C_t(u), u \in [u_i, u_{i+1}]$. C_t^i is denoted the* associated curve *of edge e_i.*

Definition 3.4. *Consider quadrilateral Q having vertices $V_j^Q \in \mathcal{V}_p, j = 0..3$, and let edge $e_j = (V_j^Q, V_{(j+1) \bmod 4}^Q) \in Q$. The* associated surface *of Q, $S_a(Q)$ is a tensor product surface defined by the Boolean sum operation between the following four curves $C_t^j, j = 0..3$:*

$$C_t^j = \begin{cases} \textit{the associated curve of } e_j, \textit{ if } e_j \textit{ is an edge in } P, \\ \\ \textit{linear edge } e_j, \textit{ otherwise }. \end{cases} \tag{1}$$

Given some weight function $W(Q, P)$ that accepts a quadrilateral (or a triangle) Q from a representative polygon P, and returns a scalar weight value, we define a weight function $\mathcal{W}$ of quadrangulation, $\mathcal{Q}$, of P as following:

$$\mathcal{W}(\mathcal{Q}, P) = \sum_{Q_i \in \mathcal{Q}} W(Q_i, P). \tag{2}$$

Each quadrangulation, $\mathcal{Q}$, of P which consists of a set of quadrilaterals $Q_i \in \mathcal{Q}$ tiling the interior of P, defines uniquely one set of associated surfaces $\{S_a(Q_i)\}$ that tiles S_t. We find the minimal quadrangulation of the curve C_t by finding the quadrangulation that minimizes $\mathcal{W}$ on the representative polygon P of C_t.

W can be designed such that the desired properties of the resulted quadrilateral surfaces are reflected in minimizing $\mathcal{W}$. We propose the following weight function: Let $e_j, j = 0..3$ be the edges of Q, and $A(Q)$ and $M(Q)$ be the area and the perimeter of Q, respectively. Further, let $J_{min}(S_a(Q))$ and $J_{max}(S_a(Q))$ be the minimal value and the maximal value of the determinant of the Jacobian of $S_a(Q)$, respectively. Then:

$$W(Q, P) = \begin{cases} \infty, \ Q \ \text{intersects edge } e_i \text{ of } P, e_i \notin Q, \\ \infty, \ Q \ \text{is self-intersecting}, \\ \infty, \ J_{min}(S_a(Q)) J_{max}(S_a(Q)) < 0, \\ \dfrac{J_{max}(S_a(Q))}{J_{min}(S_a(Q))} \left(\alpha A(Q) + \beta M(Q) + \gamma \dfrac{\max_i\{arc\_length(e_i)\}}{\min_j\{arc\_length(e_j)\}} \right), \\ \qquad\qquad i, j = 0..3, \quad \alpha, \beta, \gamma \in I\!R^+, \quad \text{otherwise}. \end{cases} \tag{3}$$

Finding $J_{min}(S_a(Q))$ and $J_{max}(S_a(Q))$ can be done, as before, by symbolically computing the determinant of the Jacobian of $S_a(Q)$, $|J(u, v)| = \left| \frac{\partial S_a}{\partial u} \times \frac{\partial S_a}{\partial v} \right|$ as a spline function. The proposed weight function in Equation (3) highly penalize quadrilaterals with invalid associated surfaces (having self intersections, etc.). For valid surfaces, it promotes surfaces with uniform Jacobian and less degenerate quadrilaterals. However, other weight functions that are less sensitive to invalid associated surfaces can be clearly provided as well, such as conformity considerations.

The complete weight function based quadrangulation algorithm is described in Algorithm 5. The algorithm applies the quadrangulation algorithm for polygons, *PolyQuadrangulate(P,W)*, that is described in Algorithm 4. There could be a case where no quadrangulation of P is free of invalid quadrilaterals. In this case, we generate a tighter representative polygon by adding more samples from C_t to P along the boundaries of the invalid quadrilaterals that lies on C_t, and apply the algorithm until all quadrilaterals are valid, see Figure 5 for an example.

Note that if the minimized weight function allows invalid quadrilaterals, then step 9 in Algorithm 5 is unnecessary, and Algorithm 5 stops after one iteration. However, depending on the weight function W, there could be a case where step 9 in Algorithm 5 might lead to unbounded number of iterations. In this case, we stop after a fixed number of iterations, and return the best minimal quadrangulation found. Patches with failing flipping Jacobian signs can then be handed-in to the line-sweep algorithm (Section 3.1.1) to ensure regularity. In the examples presented in the work, no such failing cases were observed. The

Algorithm 5 : Weight function based algorithm

Input:
C_t, a closed simple planar curve;
$W(Q, P)$, weight function for quadrilateral (or triangle) Q;
Output:
$\mathcal{Q}$, a set of freeform planar quadrilaterals, covering the interior of C_t;
Algorithm:

```
1: P := A representative polygon of C_t;
2: repeat
3:    InvalidJacobian := FALSE;
4:    Q := ∅;
5:    (Q_P, R) := PolyQuadrangulate(P,W); // Algorithm 4
6:    for all Q_i ∈ Q_P do
7:       S_a(Q_i) = Parameterize Q_i into planar patch using
            Boolean Sum;
8:       Q := Q ⋃ {S_a(Q_i)};
9:       if J_min(S_a(Q_i)) J_max(S_a(Q_i)) < 0 then
10:         InvalidJacobian := TRUE;
11:         {Q_i^s} := Additional finer samples of the outer
               boundaries of Q_i;
12:         P := P ⋃ {Q_i^s};
13:      end if
14:   end for
15: until InvalidJacobian = FALSE
16: return Q;
```

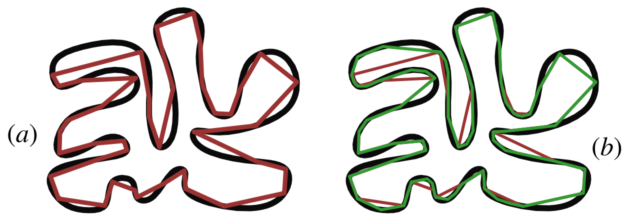

Figure 5: Iterations, in Algorithm 5, of updating the representative polygon, P, of a curve. (a) First iteration, the original curve in black and the representative polygon in red. (b) Second and final iteration, the original curve in black, the first representative polygon in red and a tighter, refined, updated representative polygon in green.

number of iteration needed for all presented examples was less than twenty, and since our set limit to switch to the line-sweep alg. was a hundred, we never switched.

The recursive algorithm, as described in Algorithm 4, has an exponential complexity with respect to n. However, we can find the minimal quadrangulation with a polynomial complexity of $O(n^4)$ by utilizing a dynamic programming approach as in [28]. This is achieved by iterating on all the sub-polygons of P from the smallest to the largest, and keeping additional memory to store the minimal weight for each such sub polygon.

Lemma 3.2. *The number of quadrilaterals that tiles a planar polygon P having n vertices is $(n-2)/2$.*

Lemma 3.2 is easily deduced from Euler's formula for planar closed graphs [29]. From Lemma 3.2, it follows that if P is tiled entirely by quadrilaterals, the number of vertices must be even. Thus, if the number of the vertices is odd, at least one triangle will be in the result. In order to avoid triangles as much as possible, we make sure the number of samples of the trimming curve, C_t, is even, and we assign a very large weight for singular rectangular patches (such as triangles).

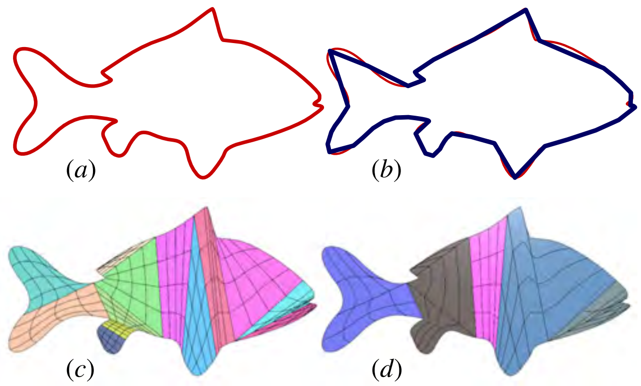

Figure 6: Steps of quadrangulation of a curve in the shape of a fish using the minimal weight algorithm: (a) The input curve. (b) The representative polygon. (c) Result of the minimal weight algorithm before the merge (13 quadrilaterals). (d) Final result after applying the merge (5 quadrilaterals).

3.2. Merging adjacent domain patches

The sampling scheme in the process of building the representing polygon doesn't guarantee optimal number of quadrilaterals. Further, some samples are introduced due to the refining step, in an attempt to avoid invalid quadrilaterals (see step 11 in Algorithm 5), and there might be unnecessary inner edges connecting unneeded samples. These edges can be removed and each two patches sharing a common edge that spans their entire domain, can be merged into a single patch. See, for example, Figures 6 and 7. This merging post-process is applied for both variations of the quadrangulation algorithms.

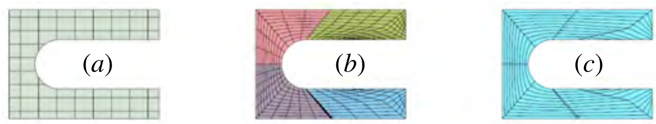

Figure 7: Merging quadrilaterals of a surface from the wrench model from Figure 11: (a) The original trimmed surface. (b) The minimal weight quadrangulation result before the merge (4 quadrilaterals). (c) After the merge, only one quad results.

We apply a simple greedy merge algorithm, that keeps merging neighboring patches, while possible. Even with such a simple merge algorithm, we could reduce the number of quadrilaterals by around 50%. (Again, see Figures 6 and 7, and also Tables 1 and 2). One should notice that the merge process decreases the number of patches, however, it typically introduces C^1 discontinuities along the merged edges.

3.3. Lifting the quadrilaterals to Euclidean space via Surface-Surface Composition

For each quadrilateral $Q_i(r,t) = (u_i(r,t), v_i(r,t)), i = 1, .., k$, generated in the algorithms described in Sections 3.1 and 3.2, we precisely construct a tensor product surface $S_{Q_i} = S(Q_i(r,t)) = S(u_i(r,t), v_i(r,t))$, where $S(u,v)$ is the tensor product surface of the trimmed-surface S_t, using surface-surface composition [6, 7, 8]. Note S is a Bézier surface at this time. The set $\mathcal{S}_{Q_i}, i = 1, ..., k$ precisely tiles and covers the original trimmed surface, $S_t(u, v)$, but consists of only tensor product patches. Hence, for example, a tight bounding box for S_t can be derived by computing the tight bounding box of the set $\mathcal{S}_{Q_i}$. More importantly, the integration over the original trimmed surface $S_t(u, v)$ can be reduced to a precise integration over a set of tensor product patches, $\mathcal{S}_{Q_i}$.

4. Results

All algorithms were implemented using the IRIT solid modeler framework [30] and were tested on a Macbook-Pro i7 2.7Ghz machine, running Window 7 64 bit. Extreme values of the Jacobian's determinant, line-curve intersection points, extreme-x, and extreme-curvature points, operations which required solving polynomial and rational equations, were all computed using the IRIT multivariate solver [26, 31].

We start with more 2D examples of complex (trimming) curves, processed by the two presented quadrangulation algorithms: the star curve in Figure 8, and the animal curves in Figures 9 and 10. Among all presented examples in this work, both polynomial and rational trimmed surfaces and curves were found, with orders up to five.

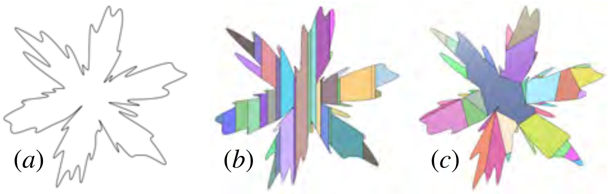

Figure 8: Star curved shape (a B-spline curve of order 3 and 100 control points): (a) The (trimming) curve. (b) Result of the line-sweep algorithm (51 tensor product surfaces). (c) Result of the minimal weight algorithm (41 tensor product surfaces).

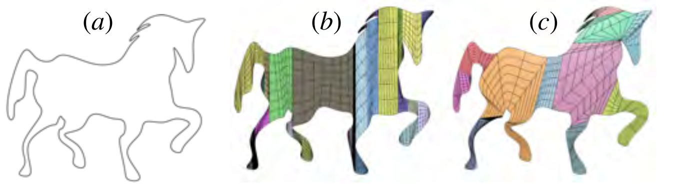

Figure 9: Horse shape (a B-spline curve of order 4 and 96 control points): (a) The (trimming) curve. (b) Result of the line-sweep algorithm (31 tensor product surfaces). (c) Result of the minimal weight algorithm (13 tensor product surfaces).

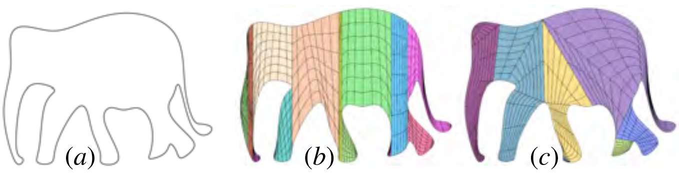

Figure 10: Elephant shape (a B-spline curve of order 4 and 54 control points): (a) The (trimming) curve. (b) Result of the line-sweep algorithm (15 tensor product surfaces). (c) Result of the minimal weight algorithm (7 tensor product surfaces).

The conversions to tensor product surfaces of two moderately complex 3D solid models from two different modeling environments, consisting of trimmed surfaces, are presented in Figures 11 and 12. Each (trimmed or tensor product) surface is painted in a different color with isoparametric curves so one can follow the established parametrizations. Clearly, the achieved parametrization is not always as appealing as one might hope for, and striving for improved parametrization can be a worthy consideration, depending on the application in hand. However, if this conversion for tensor product is toward precise integration, then the regularity of the parametrization is all that is required.

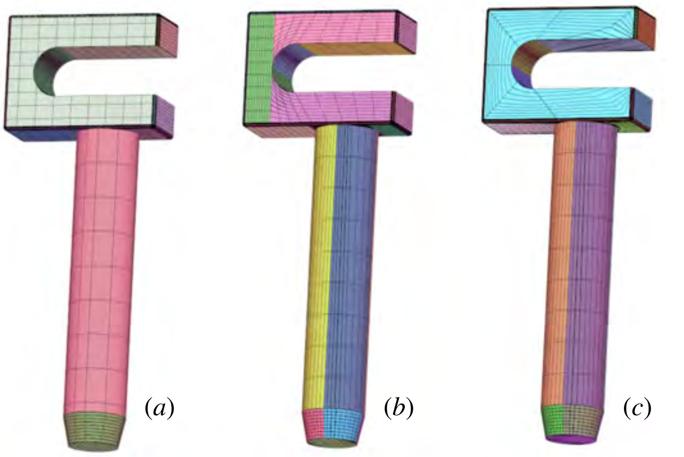

Figure 11: Wrench model: (a) The original model (38 trimmed surfaces). (b) Result of the line-sweep algorithm (53 tensor product surfaces). (c) Result of the minimal weight algorithm (45 tensor product surfaces).

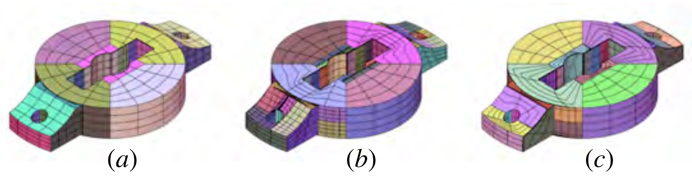

Figure 12: Solid model: (a) The original model (63 trimmed surfaces). (b) Result of the line-sweep algorithm (151 tensor product surfaces). (c) Result of the minimal weight algorithm (97 tensor product surfaces).

Figure 13 demonstrates the effect of different weight functions, on three different curves. In order to better emphasize the effect of the weight function, the results presented in Figure 13 are before applying the merge process. All the weight functions that we discuss here assign infinite weight value to invalid (self intersecting, etc.) patches. In the left column of Figure 13, the weight function used is: $W(Q) = Perimeter(Q)/\sqrt{Area(Q)}$; a weight function that assigns large weights to patches with bad aspect ratios. Indeed, this first column has almost no long and skinny patches. The weight function used in the middle column is: $W(Q) = \left\langle \frac{\partial S_a(Q)}{\partial u}, \frac{\partial S_a(Q)}{\partial v} \right\rangle^2$; a weight function that promotes conformality, and assigns smaller weight value to a mapping that preserves angles between iso-lines. More patches with close to orthogonal u and v iso-lines can be observed. Finally, the weight function used in the right column is: $W(Q) = J_{max}(S_a(Q))/J_{min}(S_a(Q))$; a weight function that promotes patches with uniform Jacobian. More fairly rectangular patches can be seen on this right column.

The presented algorithms were capable of handling complex solid models as well, and examples include a Sewing Ma-

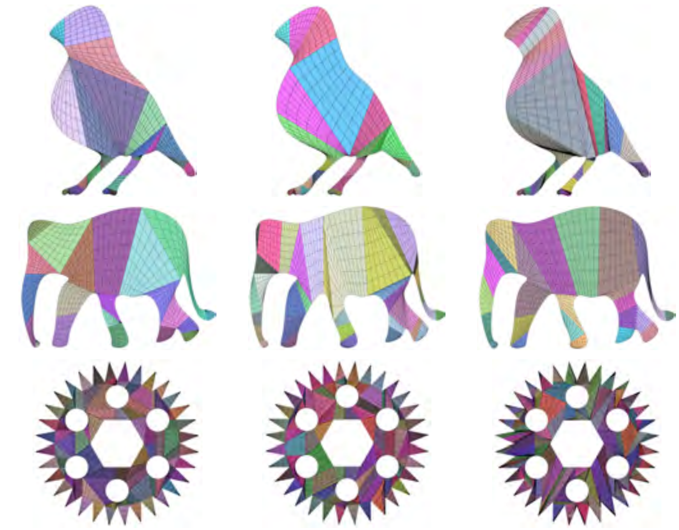

Figure 13: Three examples of three different weight functions in each. Left to right: Minimizing $Perimeter/\sqrt{Area}$, Minimizing $\left\langle \frac{\partial S}{\partial u}, \frac{\partial S}{\partial v} \right\rangle^2$, Minimizing J_{max}/J_{min}.

chine, in Figure 14, with 327 trimmed surfaces, a coffee machine, in Figure 15, with 927 trimmed surfaces and a Lawnmower, in Figure 16, with 3779 trimmed surfaces. Figure 1 exemplifies the processing of a complex trimming curve, in the shape of a teeth wheel, from the Sewing Machine in Figure 14. The weight function in Equation (3), was using, for all the examples presented in this paper, $\alpha = 0.7, \beta = 0.05$, and $\gamma = 0.1$, unless stated otherwise.

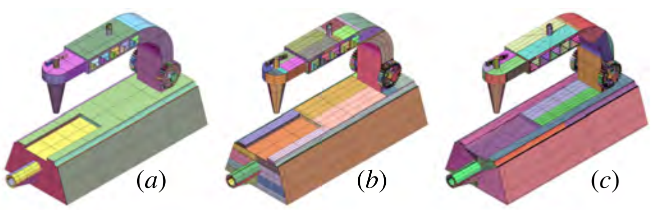

Figure 14: Sewing machine model: (a) The original model (327 trimmed surfaces). (b) Result of the line-sweep algorithm (882 tensor product surfaces). (c) Result of the minimal weight algorithm (693 tensor product surfaces).

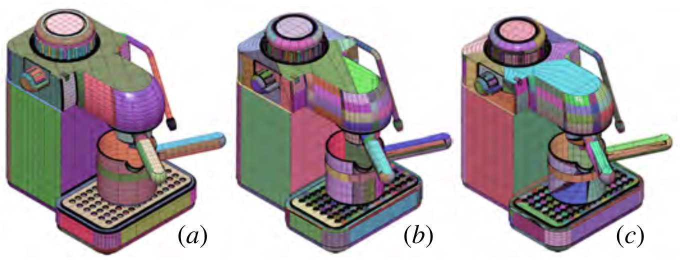

Figure 15: Coffee machine model: (a) The original model (927 trimmed surfaces). (b) Result of the line-sweep algorithm (3047 tensor product surfaces). (c) Result of the minimal weight algorithm (3152 tensor product surfaces).

Tables 1 and 2 provide more details and statistics. The tables provide the number of input trimmed surfaces for each presented model, the number of output patches (quadrilaterals) the quadrangulation method produced, before and after the merge process, and the number of patches that are singular in at least

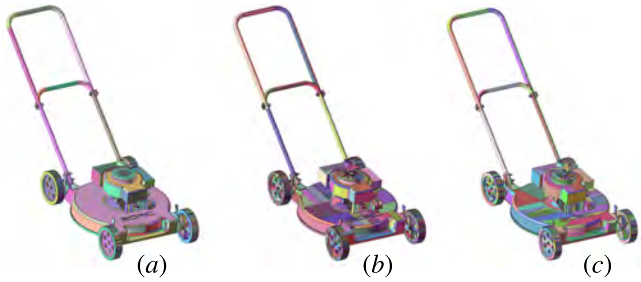

Figure 16: Lawnmower model: (a) The original model (3779 trimmed surfaces). (b) Result of the line-sweep algorithm (8246 tensor product surfaces). (c) Result of the minimal weight algorithm (8495 tensor product surfaces).

one point on their boundary. No patch in any of the presented results is singular in an interior location. In the line sweep-algorithm, singularity can be introduced in start and end events where the generated quadrilateral is triangular. And in the minimal weight algorithm a quadrilateral that is composed of two segments of the trimming curve that share a C^1 end point will be singular at that point. Hence, since we dont add interior Steiner points, singularities frequently happen at the boundaries, having (potentially) $J_{min} = 0$ at some points on the boundaries. The minimal weight algorithm can use a weight function that penalizes such cases and avoids as much as possible quadrilaterals composed of adjacent C^1 segments. However, we haven't use such weight function as we focus more on IGA applications, which require no singularities at the interior.

Also provided in these tables, are computation times. As can be seen in Tables 1 and 2, the minimal weight quadrangulation algorithm typically resulted in less tensor product surfaces compared to the line-sweep quadrangulation algorithm. However, the line-sweep algorithm is less time consuming. The total times stated in Tables 1 and 2 include the entire processing as portrayed by Algorithm 1, including the quadrangulations and surface-surface composition. Table 1 also provides the times for the surface-surface composition, for the line-sweep algorithm. As can be seen, the surface-surface compositions consume between 5% to 25% of the time, in this case, while the line-sweep quadrangulation consumes most of the computation time. On the other hand, for the minimal weight quadrangulation algorithm, the computational costs of the compositions are negligible.

As stated, we are able to precisely compute differential and integral properties over the tensor products, computations that are far more difficult when the trimmed surfaces are provided. Herein, we briefly show how to compute the volume of the object and its precise bounding box. Our input consists of the models in Figure 17. Since we used symbolic integration, the geometry must be (piecewise) polynomial. Hence, arcs and circles were approximated using piecewise polynomials to an accuracy of $\sim 10^{-3}$. The volume of the quarter of a torus in Figure 17 (a) can be computed analytically. The analytic value is 0.493480, whereas the integration over geometry converted to tensor product surfaces yielded the result of 0.493757 and 0.493757 (using our two portrayed quadrangulation methods). The last result is well within the arc approximation and in precise agreement between the two presented quadrangulation algorithms.

While we do not know the precise volume of the model in

Model	#Surfaces in original model	Line-sweep algorithm				
		#Patches Before Merge	#Patches After Merge	#Singular on Boundary	Time (Sec.) Total	Time (Sec.) Composition
Star (Figure 8)	1	51	51	50	0.0168	0.0023
Horse (Figure 9)	1	34	31	28	0.0165	0.0007
Elephant (Figure 10)	1	18	15	12	0.00796	0.00016
Wrench (Figure 11)	38	53	53	8	0.031	0.0099
Solid (Figure 12)	63	159	151	25	1.372	0.25
Sewing machine (Figure 14)	327	917	882	205	3.681	0.702
Coffee machine (Figure 15)	927	3397	3047	570	2.869	0.28
Lawnmower (Figure 16)	3779	9945	8246	2534	4.613	1.168

Table 1: Number of generated quadrilaterals and running time for the line-sweep algorithm. Compare with Table 2.

Model	#Surfaces in original model	Minimal weight algorithm			
		#Patches Before Merge	#Patches After Merge	#Singular on Boundary	Time (Sec.) Total
Star (Figure 8)	1	76	41	36	238
Horse (Figure 9)	1	46	13	10	38.8
Elephant (Figure 10)	1	23	7	7	1.48
Wrench (Figure 11)	38	103	45	9	5.85
Solid (Figure 12)	63	240	97	14	25.4
Sewing machine (Figure 14)	327	1543	693	161	73.7
Coffee machine (Figure 15)	927	7978	3152	668	4834
Lawnmower (Figure 16)	3779	23062	8495	3069	3009

Table 2: Number of generated quadrilaterals and running time for the minimal weight algorithm. Compare with Table 1.

Figures 17 (b), we can again compare the results of the two quadrangulation methods. For Figure 17 (b), the computed volumes are 2.342239 and 2.342241, respectively, for the two quadrangulation variations.

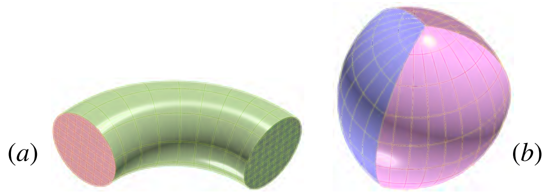

Figure 17: Two examples of trimmed surfaces models used for precise evaluations of differential and integral properties, via a conversion to tensor product surfaces.

The ability to compute precise bounding boxes to tensor product surfaces (by examining the $x-$, $y-$, and $z-$extrema in the interior of the tensor product patches and on their boundary curves) allows one to compute precise bounding boxes to the converted geometry in hand. Figure 18 (a) shows the bounding box computed for the trimmed surfaces' model (after clipping the tensor product surface of the trimmed surface to the 2D bounding box of the trimming curves) by examining the coefficients of the clipped surfaces. Figures 18 (b) and (c) show the precise bound boxes that result (by examining the $x-$, $y-$, and $z-$extrema, using differential analysis and computed with the aid of the converted model, using the two quadrangulation variants, and consisting solely of tensor product surfaces.

As stated earlier, aiming at IGA applications, we show, in Figure 19, IGA simulation results of large deformation elasticity analysis, utilizing the proposed untrimming approach, on a 3D trimmed object. The untrimming into tensor product B-spline patches and then to Bezier patches (as required by the IGA analysis) is shown in Figure 19 (b) whereas one 2D solution is presented in Figure 19 (c). These Bezier patches are then all rotated in space to form the volume of revolution and enable the 3D analysis presented in (d). See also Acknowledgments.

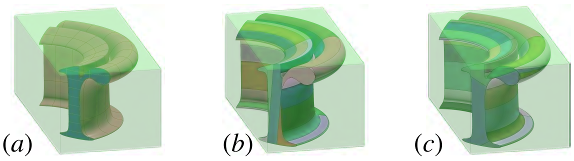

Figure 18: The bounding box of a sweep model of a letter 'r' is computed using the original trimmed surfaces, examining the control points of the tensor product surface that was clipped to the bounding box of the trimming curves (a). A tight bounding box is computed by deriving the precise $x-$, $y-$, and $z-$extrema of the tensor product surfaces, computed using the line-sweep quadrangulation (b) and the minimal weight algorithm (c).

It is interesting to examine the orders of the resulting surfaces. Figure 20 presents the orders' distribution of (trimmed) surfaces in the input and the output (tensor product) surfaces, for both variations of quadrangulations. As expected and due to the surface-surface composition, the orders of the surfaces on the output are higher. Further, since the Line-sweep quadrangulation algorithm uses ruled surfaces and the minimal weight quadrangulation algorithm uses Boolean sum, the orders of the resulting surfaces of the later can be higher. In some of the examples, as can be observed in extreme cases in Figure 20, the Boolean sum and the surface-surface composition can result in

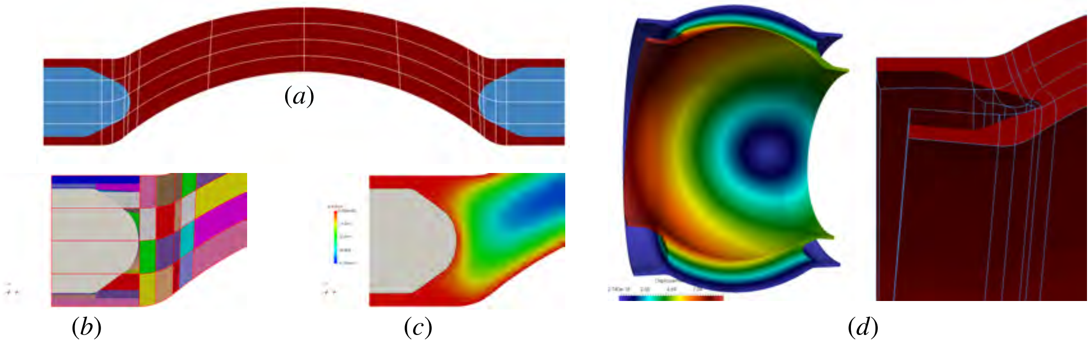

Figure 19: Large deformation elasticity analysis on a trimmed object utilizing IGA with the proposed untrimming approach. (a) A cross section of the object consisting of one trimmed surface (in dark red), trimmed at both ends (blue zones). (b) Untrimming results of the cross section surface (showing one end), resulting in only Bezier patches, ready for IGA analysis. (c) The 2D IGA solution on the cross section. (d) The 3D solution of the full model, which is a volume of revolution of the 2D cross section Bezier surfaces. See also Acknowledgments.

higher degrees, especially when rational trimmed surfaces and trimming curves are involved.

Finally, in `https://sites.google.com/site/untrimming/` IGES files of some of the presented models, as input trimmed surfaces and after converted to tensor products by both presented algorithm, can be found.

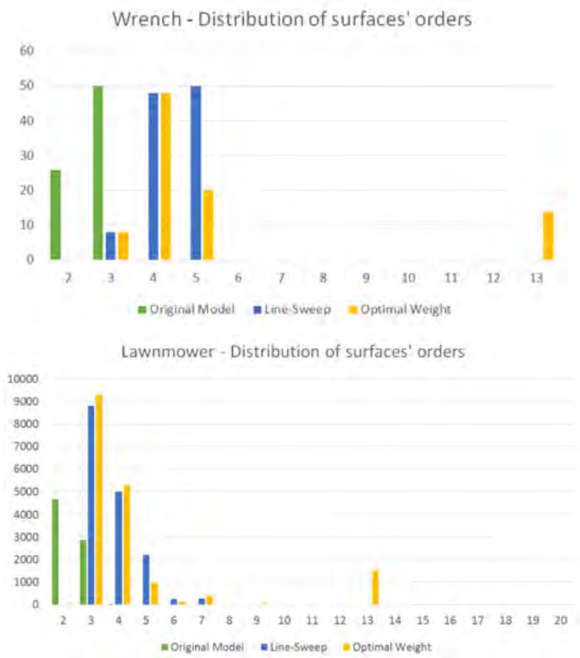

Figure 20: The distribution of orders in the input and output surfaces for the Wrench model in Figure 11 (top) and for the Lawnmower in Figure 16 (bottom). Note we count instances, having two instances of orders per surface, being a bivariate function.

5. Conclusion and future work

In this paper, we introduce methods for converting a model consisting of trimmed-surfaces into a set of tensor-product surfaces. The algorithm is robust and preserves the precision of the trimmed surfaces. Two variations of the algorithm are proposed for the quadrangulation stage. The first is an efficient line-sweep based approach, and the second allows user control over the result by minimizing a given weight function.

There is room for improvement in the merge algorithm, in terms of reducing the number of total patches by utilizing some global optimization process. In addition, the number of generated patches can be further reduced by merging back adjacent quadrilaterals that belong to the same original trimmed B-spline surface but were divided to different trimmed Bézier surfaces, due to internal knots or holes.

The optimal orientation of the swept line in the line-sweep based quadrilateral generation algorithm, for generating minimal number of patches is a degree of freedom that will be interesting to explore. In the minimal weight algorithm, there might be triangular patches in the result, while we reduce their frequency by assigning a large penalty for triangles. However, in some cases, depending on the weight function, triangles can't be avoided. Though each triangle can be split into three quadrilaterals, we strive to have only quadrilaterals in the result, without such splits. A better curve sampling scheme might result in less triangles and fewer number of patches in general.

In this work, we only ensured that a trimmed surface will be faithfully and precisely reconstructed using a set of tensor product surfaces. If cracks (black holes) exist between adjacent trimmed surfaces, that problem will persist. Stitching algorithms, while not part of this work, will complement the algorithms presented here, and will make them complete.

Finally, the adaptation of the presented conversion methods toward the untrimming of trimmed-volumes [2], is highly desired. Going up a dimension, from planar domains to volumetric ones, is a major challenge. Yet, this need for volumetric integration is already here, toward the precise IGA computation.

6. Acknowledgments

The Lawnmower (Figure 16), the Coffee Machine (Figure 15), and the Wrench (Figure 11), are from the Design Repository (http://edge.cs.drexel.edu/repository/). The freeform animal curves in Figures 9 and 10 were created by In-Kwon Lee and Myung Soo Kim, Postech, Korea. The Sewing machine in Figure 14 was created by Miriam Band and Oded Messer, Technion. The

Solid example (Figure 12) is from the IRIT modeling environment. All data was provided in IGES form.

Figure 19 was created in collaborations with Pablo Antolin (EPFL Lausanne), Annalisa Buffa (EPFL Lausanne and IMATI-CNR Pavia), Massimiliano Martinelli (IMATI-CNR Pavia); Giancarlo Sangalli (University of Pavia and IMATI-CNR Pavia).

References

[1] Cohen E, Riesenfeld R, Elber G. Geometric Modeling with Splines: An Introduction. A. K. Peters, Ltd.; 2001.

[2] Massarwi F, Elber G. A B-spline based framework for volumetric object modeling. Computer-Aided Design 2016;78:36 – 47. SPM 2016.

[3] Satoh T, Chiyokura H. Boolean operations on sets using surface data. In: Proceedings of the First ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications. SMA '91; New York, NY, USA: ACM; 1991, p. 119–26.

[4] Thomas SW. Set Operations on Sculptured Solids. Technical report; University of Utah, Department of Computer Science; 1986.

[5] Cottrell JA, Hughes TJ, Bazilevs Y. Isogeometric analysis: toward integration of CAD and FEA. John Wiley & Sons; 2009.

[6] Elber G, Kim MS. Modeling by composition. Computer-Aided Design 2014;46:200 –4. 2013 {SIAM} Conference on Geometric and Physical Modeling.

[7] DeRose T, Goldman R, Hagen H, Mann S. Functional composition via blossoming. ACM Transactions on Graphics 1993;12(2):113–35.

[8] Elber G. Free form surface analysis using a hybrid of symbolic and numeric computation. Ph.D. thesis; University of Utah; 1992.

[9] Bommes D, Zimmer H, Kobbelt L. Mixed-integer quadrangulation. ACM Transactions On Graphics (TOG) 2009;28(3):77.

[10] Bommes D, Campen M, Ebke HC, Alliez P, Kobbelt L. Integer-grid maps for reliable quad meshing. ACM Transactions on Graphics (TOG) 2013;32(4):98.

[11] Ebke HC, Campen M, Bommes D, Kobbelt L. Level-of-detail quad meshing. ACM Transactions on Graphics (TOG) 2014;33(6):184.

[12] Bommes, David and Lévy, Bruno and Pietroni, Nico and Puppo, Enrico and Silva, Claudio and Tarini, Marco and Zorin, Denis . Quad-Mesh Generation and Processing: A Survey. In: Computer Graphics Forum; vol. 32. 2013, p. 51–76.

[13] Jingjing Shen and Jiri Kosinka and Malcolm A. Sabin and Neil A. Dodgson . Conversion of trimmed {NURBS} surfaces to Catmull-Clark subdivision surfaces. Computer Aided Geometric Design 2014;31(7-8):486 – 498.

[14] Schollmeyer A, Fröhlich B. Direct trimming of nurbs surfaces on the gpu. ACM Trans Graph 2009;28(3):471–9.

[15] Martin W, Cohen E, Fish R, Shirley P. Practical ray tracing of trimmed nurbs surfaces. Journal of Graphics Tools 2000;5(1):27–52.

[16] Balázs Á, Guthe M, Klein R. Efficient trimmed nurbs tessellation. Journal of WSCG 2004;12(1):27–33.

[17] Piegl LA, Richard AM. Tessellating trimmed nurbs surfaces. Computer-Aided Design 1995;27(1):16–26.

[18] YingLiang M, Hewitt T. Adaptive tessellation for trimmed nurbs surface. Citeseer. Retrieved from http://wscg.zcu.cz/wscg2003/Papers_2003/H19.pdf; 2002,.

[19] Kahlesz F, Balázs Á, Klein R. Multiresolution rendering by sewing trimmed nurbs surfaces. In: Proceedings of the seventh ACM symposium on Solid modeling and applications. ACM; 2002, p. 281–8.

[20] Sederberg TW, Finnigan GT, Li X, Lin H, Ipson H. Watertight trimmed nurbs. ACM Transactions on Graphics (TOG) 2008;27(3):79.

[21] Hamann B, Tsai PY. A tessellation algorithm for the representation of trimmed nurbs surfaces with arbitrary trimming curves. Computer-Aided Design 1996;28:461–72.

[22] Hoschek J, Schneider FJ. Spline conversion for trimmed rational Bézier- and B-spline surfaces. Computer-Aided Design 1990;22(9):580–90.

[23] Hui K, Wu YB. Feature-based decomposition of trimmed surface. Computer-Aided Design 2005;37(8):859 –67. CAD '04 Special Issue: Modelling and Geometry Representations for CAD.

[24] Li X, Chen F. Exact and approximate representations of trimmed surfaces with nurbs and Bézier surfaces. In: Computer-Aided Design and Computer Graphics, 2009. CAD/Graphics' 09. 11th IEEE International Conference on. IEEE; 2009, p. 286–91.

[25] de Berg M, Cheong O, van Kreveld M, Overmars M. Computational Geometry Algorithms and Applications. 3rd ed.; Springer Berlin Heidelberg; 2008.

[26] Machchhar J, Elber G. Revisiting the problem of zeros of univariate scalar béziers. Computer Aided Geometric Design 2016;43:16–26.

[27] Cohen S, Elber G, Bar-Yehuda R. Matching freeform curves. Computer-Aided Design 1997;29:369–78.

[28] Klincsek G. Minimal triangulations of polygonal domains. Ann Discrete Math 1980;9:121–3.

[29] Richeson DS. Euler's gem: the polyhedron formula and the birth of topology. Princeton University Press; 2012.

[30] Elber G. Irit 11 user's manual 2015;URL: `http://www.cs.technion.ac.il/~irit`.

[31] Elber G, Kim MS. Geometric constraint solver using multivariate rational spline functions. In: Proceedings of the sixth ACM symposium on Solid modeling and applications. ACM; 2001, p. 1–10.