Are Two Curves the Same?

Diana Pekerman[1], Joon-Kyung Seong[2], Gershon Elber[3], and Myung-Soo Kim[4]

[1]Technion-Israel Institute of Technology, pdiana@cs.technion.ac.il
[2] Seoul National University, swallow@3map.snu.ac.kr
[3]Technion-Israel Institute of Technology, gershon@cs.technion.ac.il
[4]Seoul National University, mskim@cse.snu.ac.kr

ABSTRACT

We present an algorithm for answering the following fundamental question: Given two arbitrary (piecewise) polynomial curves, are they the same? This basic CAGD question is answered by first reducing the two curves into canonical irreducible forms. This is done by reversing the processes of knot refinement, degree raising, and composition. The two curves are then compared in their irreducible forms and their shared domains, if any, are identified. The ability to answer this fundamental identity question will be a boon for numerous applications. In this paper, we demonstrate a few such applications. The algorithm allows one to identify two boundary curves (shared as a common seam) between two different surfaces as an identical curve (or not) even when they are represented differently. Moreover, we show that reparameterization is insecure as a watermarking method, which invalidates the proposal of [8].

Keywords: Polynomials; rationals; composition; decomposition; knot refinement; knot removal; degree-raising; degree-reduction; water-marking; curve matching.

1. INTRODUCTION

There are several known ways of modifying the representation of a piecewise polynomial/rational regular curve, $C(t)$, while preserving its geometry or trace:

- Degree Raising [1]. A piecewise polynomial/rational curve of degree d can always be raised and represented as a curve of higher degree $d + k$, $k = 0$.
- Refinement [1]. Every piecewise polynomial/rational curve can be refined, i.e., new knots can be inserted into the curve, without affecting its shape. However, these new knots introduce potential discontinuities into the geometry.
- Composition [2], [5]. Let $t(r)$ be an allowable change of parameter. Then, the trace of $\hat{C}(r) = C(t(r))$ is identical to that of $C(t)$, albeit in a possibly different domain or speed.

It is quite well known that both degree raising and refinement are reversible. That is, given a curve $C(t)$ known to be of degree d, which is represented as a curve of degree $d + k$, $k > 0$, one can deduce its original representation in degree d. One simple way to achieve this, though not necessarily the most stable one, is to convert the representation to monomial form as $C(t) = \sum_{i=0}^{d+k} a_i t^i$ and purge all the coefficients a_i, for $i > d$, that are zero. Then, convert the curve back to the representation based on its original basis functions. More on degree reduction can be found, for example, in [9].

Similarly, the process coined "knot removal" [7] attempts to detect knots whose removal has little or no effect over the shape of the curve. Remove knot t_i and examine the best/closest shape one can find to the original curve $C(t)$, under a certain norm, in the subspace that excludes knot t_i. If the new shape is identical to $C(t)$, the curve $C(t)$ resides in the subspace that excludes t_i, since the knot t_i only introduces a potential discontinuity, and therefore, it can be removed.

The third way to modify the representation while preserving the shape is via functional composition. The composition function modifies the speed of the curve or its domain but not the curve trace. In fact, when the composition is made using an identity function, that is $t(r) = r$, the composition is reduced to a degree raising operation. Let the degree of $t(r) = r$ be k, e.g., $t(r) = r[(1-r)+r]^{k-1}$, and let the degree of $C(t)$ be d. Then, the

composition is reduced to a simple degree raising, with $\hat{C}(r) = C(t(r))$ being of degree dk. It is interesting to note that another simple way of achieving a degree raising effect is by multiplying a constant one of degree l. Namely, let $p(t) = 1$ be a (constant) polynomial of degree l, e.g., $p(t) = [(1-t)+t]^l$. Then, $D(t) = C(t)\,p(t)$ is a new curve of degree $d + l$, but with exactly the same shape as the curve $C(t)$. In this paper, we assume only allowable changes of parameters, $t(r)$, in the functional compositions; that is, $t'(r) > 0$, $\forall r$.

Reversing the composition operation on polynomials, or *decomposition* of polynomials, is a complex task; yet it is a feasible operation as we demonstrate it in this paper. Symbolic processing software packages, such as Maple [3] or Mathematica [10], support the composition functionality as a built-in procedure. They do not, however, support the reverse operation of the composition. Given a composed function $H(x)$, we seek to find its decomposition, if any, as $H(x) = f(g(x))$. In other words, we find the original functions, $f(x)$ and $g(x)$, if they exist.

The fact that symbolic manipulation programs provide no such reverse capability is more surprising, as function decomposition has great importance in general, and in geometric modeling, specifically. In fact, the first algorithm to decompose $H(x)$ into $f(x)$ and $g(x)$ was proposed by [5] more than 15 years ago, and it was improved thereafter by [2].

Degree raising and refinement were both previously suggested as weak watermarking techniques. It is interesting to note that the composition operation has been proposed to be a highly robust way to watermark spline models [8]. However, a clear implication of this paper invalidates the proposal of [8] – we show that composition is insecure as a watermarking method.

Another important need for identifying the similarity of two curves could be found in the way geometry is typically represented in contemporary geometric modelers. A model is typically a collection of freeform (trimmed) surfaces that are stitched together along shared seams. In many cases the two curves sharing the seam are identical geometrically but are represented differently due to the way the surfaces containing these two curves were constructed. For example, to construct a ruled surface between a quadratic and a cubic curve, the quadratic is artificially degree-raised to be a cubic. Having two curves sharing a seam represented differently makes cross-seams processing difficult. For example, in order to create a water-tight tessellation of the geometry, both surfaces, along this shared seam, must be sampled in exactly the same way. Knowing that these two curves are indeed the same curve can greatly simplify this process.

The rest of this paper is organized as follows. In Section 2, we present our approach to the functional decomposition problem, and outline the complete algorithm, exploiting the unique structure that allows one to avoid solving a non-linear system of constraints. In Section 3, an algorithm is presented that brings all these functionalities into play in a single identity testing algorithm. In Section 4, we demonstrate a few interesting applications and examples, and finally, we conclude this paper in Section 5.

2. FUNCTIONAL DECOMPOSITION

Consider two non-linear polynomials $f(x)$ and $g(x)$. The resulting composition of $(f \circ g)(x) = f(g(x))$ is a higher degree polynomial with many non-linear terms. In general, finding a solution to a set of non-linear equations is difficult. In our case, these equations are well structured, and we will employ this simple structure to efficiently deduce the decomposition, if it exists. Moreover, its unique structure is exactly the same in projective space when $f(x)$ is rational.

The key observation that makes the functional decomposition tractable can be summarized as follows: In the composition of two polynomials $f(x)$ and $g(x)$, the lowest degree terms reveal the structure of $f(x)$ and the highest degree terms expose the structure of $g(x)$. The result of the composition has many high degree terms. However, and being closely related to convolution, many of these terms assume a simple form at the two ends of this composition process. In other words, the terms with the highest and lowest degrees are the simplest. In Section 2.1, we provide additional corroboration of our claim that no generality is lost by employing the following canonical forms of $f(x)$ and $g(x)$, for a general composition $H(x) = f(g(x))$,

$$\begin{aligned} f(x) &= x^m + b_{m-1}x^{m-1} + \cdots + b_1 x + b_0, \\ g(x) &= x^k + c_{k-1}x^{k-1} + \cdots + c_1 x. \end{aligned} \tag{1}$$

By exploiting this canonical form, we could greatly simplify the composition formulation and its structure analysis. When the two functions in Eqn. (1) are composed, we end up with a polynomial of degree km:

$$\begin{aligned} H(x) &= x^{km} + a_{km-1}x^{km-1} + \cdots + a_1 x + a_0 = f(g(x)) \\ &= (x^k + c_{k-1}x^{k-1} + \cdots + c_1 x)^m + b_{m-1}(x^k + c_{k-1}x^{k-1} + \cdots + c_1 x)^{m-1} + \cdots + b_1(x^k + c_{k-1}x^{k-1} + \cdots + c_1 x) + b_0. \end{aligned}$$

Examine the coefficients of the k terms of $H(x)$ with the highest degrees. Each coefficient c_{k-i} is expressible as a function of a_{km-i} and $c_k, \cdots, c_{k-i+1}$, for $i = 1, \cdots, k-1$, where $c_k = 1$. Once the coefficients of $g(x)$ are computed in

this way, the relationship between a_i's and b_j's is linear. Each coefficient b_j is again computed as an explicit function of a_j, $c_1,\cdots,c_{k-1}$, and $b_0,\cdots,b_{j-1}$, for $j = 0,\cdots,m-1$.
The rest of this section is organized as follows. In Section 2.1, we reduce the general problem into a canonical representation to ease the solution process. The actual algorithm is presented in Section 2.2.

2.1 A Template Solution

In this section, we present an affine transformation that simplifies the procedure of decomposing a polynomial $H(x) = f(g(x))$ into two polynomials $f(x)$ and $g(x)$, if such exist. This is to justify the simplified structure of $f(x)$ and $g(x)$ in Eqn. (1).
Consider the affine transformation, which is also a linear polynomial: $l(x) = ax + b$, for some constants $a \neq 0$ and ß. Then, its inverse transformation is another linear polynomial: $l^{-1}(x) = a^{-1}(x - b)$.
Given a composite polynomial $H(x) = f(g(x))$, there are infinitely many different ways that $H(x)$ may be decomposed into f and g using the above linear polynomials:

$$H(x) = f(g(x)) = \hat{f}(\hat{g}(x)) \text{ where } \hat{f}(x) = (f \circ l^{-1})(x),\ \hat{g}(x) = (l \circ g)(x).$$

If $f(x)$ and $g(x)$ are polynomials, then $\hat{f}(x)=(f\circ l^{-1})(x)$ and $\hat{g}(x)=(l\circ g)(x)$ are also polynomials of the same degrees, respectively.
Assume that $g(x)$ is a polynomial of degree k: $g(x) = c_k x^k + c_{k-1}x^{k-1} + \cdots + c_1 x + c_0$, where $c_k \neq 0$.
Using the linear polynomial, $l(x) = c_k^{-1}(x - c_0)$, we get $l(g(x))$:

$$\hat{g}(x) = x_k + (c_k^{-1}c_{k-1})x^{k-1} + \cdots + (c_k^{-1}c_1)x.$$

Thus, without loss of generality, we may assume that $c_k = 1$ and $c_0 = 0$. We will now assume that $f(x)$ is a polynomial of degree m and that $g(x)$ is given in the special form, with $c_k = 1$ and $c_0 = 0$:

$$f(x) = b_m x^m + b_{m-1}x^{m-1} + \cdots + b_1 x + b_0,$$

$$g(x) = x^k + c_{k-1}x^{k-1} + \cdots + c_1 x,$$

for some $b_m \neq 0$. Their composition $H(x) = f(g(x))$ is a polynomial of degree km and it can be represented as follows:

$$\begin{aligned} H(x) &= a_{km}x^{km} + a_{km-1}x^{km-1} + \cdots + a_1 x + a_0 = f(g(x)) \\ &= b_m(x^k + c_{k-1}x^{k-1} + \cdots + c_1 x)^m + b_{m-1}(x^k + c_{k-1}x^{k-1} + \cdots + c_1 x)^{m-1} + \cdots + b_1(x^k + c_{k-1}x^{k-1} + \cdots + c_1 x) + b_0. \end{aligned} \quad (2)$$

Comparing the terms of degree km, we note that $a_{km} = b_m$. Subdividing the above equation by setting $a_{km} = b_m$, we may assume that both $H(x)$ and $f(x)$ have 1 as the coefficient of their leading terms. Therefore, in the ensuing discussion of this paper, we assume that,

$$H(x) = x^{km} + a_{km-1}x^{km-1} + \cdots + a_1 x + a_0,$$

$$f(x) = x^m + b_{m-1}x^{m-1} + \cdots + b_1 x + b_0,$$

$$g(x) = x^k + c_{k-1}x^{k-1} + \cdots + c_1 x,$$

and also that:

$$f(g(x)) = (x^k + c_{k-1}x^{k-1} + \cdots + c_1 x)^m + b_{m-1}(x^k + c_{k-1}x^{k-1} + \cdots + c_1 x)^{m-1} + \cdots + b_1(x^k + c_{k-1}x^{k-1} + \cdots + c_1 x) + b_0.$$

2.2 Recurrence Formulas

This section presents recurrence formulas that can be used in computing the coefficients c_{k-i} and b_j, recursively.
We first define the coefficients $A[l,i]$, $(l = 1,\cdots,m;\ i = 0,\cdots,kl-l)$ as follows:

$$(c_k x^k + c_{k-1}x^{k-1} + \cdots + c_1 x)^l = \sum_{i=0}^{kl-l} A[l,i]x^{kl-i}, \text{ where } c_k = 1.$$

Note that

$A[1,i] = c_{k-i}$, for $i = 0,\cdots,k-1$;

$A[l,0] = c_k^l = 1$, for $l = 1,\cdots,m$; and

$A[m,i] = a_{km-i}$, for $i = 1,\cdots,k-1$.

Moreover, for each $l = 2,\cdots,m$, using the relation

$$\begin{aligned} (x^k + c_{k-1}x^{k-1} + \cdots + c_1 x)^l &= (x^k + c_{k-1}x^{k-1} + \cdots + c_1 x)\cdot(x^k + c_{k-1}x^{k-1} + \cdots + c_1 x)^{l-1} \\ &= (x^k + c_{k-1}x^{k-1} + \cdots + c_1 x)\cdot\left(\sum_{i=0}^{k(l-1)-(l-1)} A[l-1,i]x^{k(l-1)-i}\right). \end{aligned}$$

we can derive the following recurrence formula:

$A[l,i] = \sum_{j=0}^{i} c_{k-i+j} A[l-1, j]$, for $i = 0, \cdots, k-1$.

By induction, we can easily show that $A[l,i]$ is determined by the coefficients $c_k, \cdots, c_{k-i}$.

Using this recurrence formula, we can derive the following equation:

$$\begin{aligned} a_{km-i} &= A[m,i] = \sum_{j=0}^{i} c_{k-i+j} A[m-1, j] \\ &= c_{k-i} A[m-1,0] + A[m-1,i] + \sum_{j=1}^{i-1} c_{k-i+j} A[m-1, j] \\ &= c_{k-i} + c_{k-i} A[m-2,0] + A[m-2,i] + \sum_{j=1}^{i-1} c_{k-i+j} A[m-2, j] + \sum_{j=1}^{i-1} c_{k-i+j} A[m-1, j] \\ &= 2c_{k-i} + A[m-2,i] + \sum_{l=m-2}^{m-1} \sum_{j=1}^{i-1} c_{k-i+j} A[l, j] = (m-1)c_{k-i} + A[1,i] + \sum_{l=1}^{m-1} \sum_{j=1}^{i-1} c_{k-i+j} A[l, j] \\ &= mc_{k-i} + \sum_{l=1}^{m-1} \sum_{j=1}^{i-1} c_{k-i+j} A[l, j]. \end{aligned}$$

Consequently, we have

$$c_{k-i} = \frac{a_{km-i} - \sum_{l=1}^{m-1} \sum_{j=1}^{i-1} c_{k-i+j} A[l, j]}{m}, \text{ for } i = 1, 2 \cdots, k-1. \tag{3}$$

Note that each term $A[l, j]$ was determined by $c_k, \cdots, c_{k-i}$. Thus, c_{k-i} is completely determined by a_{km-i} and $c_k, \cdots, c_{k-i+1}$.

Once the values of $c_{k-1}, \cdots, c_1$ are determined using the above recurrence relation, each coefficient b_i can be represented as a rational expression of $c_{k-1}, \cdots, c_1$ and $b_0, \cdots, b_{i-1}$. For this purpose, we define the coefficients $B[l,i]$, $(l = 1, \cdots, m;\ i = l, \cdots, kl)$, as follows:

$(x^k + c_{k-1}x^{k-1} + \cdots + c_1 x)^l = \sum_{i=l}^{kl} B[l,i] x^i$.

Note that

$B[1,i] = c_i$, for $i = 1, \cdots, k-1$; and

$B[l,l] = c_1^l = 1$, for $l = 1, \cdots, m$.

Each coefficient $B[l,i]$ can be computed using the following equation:

$$\begin{aligned} \sum_{i=l}^{kl} B[l,i] x^i &= (x^k + c_{k-1}x^{k-1} + \cdots + c_1 x)^l = (x^k + c_{k-1}x^{k-1} + \cdots + c_1 x) \cdot (x^k + c_{k-1}x^{k-1} + \cdots + c_1 x)^{l-1} \\ &= (x^k + c_{k-1}x^{k-1} + \cdots + c_1 x) \cdot \left(\sum_{i=l-1}^{k(l-1)} B[l-1,i] x^i \right). \end{aligned}$$

A recurrence relation for $B[l,i]$ can be formulated as follows:

$B[l,i] = \sum_{j=1}^{k} c_j B[l-1, i-j]$.

Since $B[l-1, i-j] = 0$ for $i - j < l-1$ and $i - j > k(l-1)$, we have

$B[l,i] = \sum_{j=\max(1, i-k(l-1))}^{\min(k, i-l+1)} c_j B[l-1, i-j]$.

Once all the coefficients $B[l,i]$ are computed, we can represent the value of b_l, $(l = 1, \cdots, m-1)$, as a rational expression of $a_l, b_0, \cdots, b_{l-1}$, and $B[l,1], \cdots, B[l-1,l]$, where we assume $B[p,l] = 0$ if $l < p$ or $pk < l$.

From Eqn. (2), we have $a_l = b_l c_1^l + \sum_{p=1}^{l-1} b_p B[p,l]$.

Thus, in case $c_1 \neq 0$, b_l has the following rational representation:

$$b_l = \frac{a_l - \sum_{p=1}^{l-1} b_p B[p,l]}{c_1^l}.$$

Now, we consider the general case where $g(x) = x^k + c_{k-1}x^{k-1} + \cdots + c_s x^s$, that is, $c_{s-1} = \cdots = c_1 = 0$. Then we have

$a_{sl} = b_l c_s^l + \sum_{p=1}^{l-1} b_p B[p, sl]$, and

$$b_l = \frac{a_{sl} - \sum_{p=1}^{l-1} b_p B[p,l]}{c_s^l}. \tag{4}$$

The following two algorithms summarize the procedure for computing the coefficients of $f(x)$ and $g(x)$. For each divisor k of n, $(1 < k < n)$, we set $m = n/k$ and call Algorithms 1 and 2 to construct two component polynomials

$g(x)$ and $f(x)$, and test if $H(x) = f(g(x))$ holds. From the nested loops in their pseudo-codes, we can see that Algorithm 1 is always computed in $O(kn)$ time, whereas Algorithm 2 is computed in $O(n^2)$ time. This time the complexity may look relatively high. However, consider that n is the degree of the polynomial $H(x)$, which is usually smaller than 30 for all practical purposes. For a given $n < 30$, Algorithms 1 and 2 are called, at most, five times. Thus, the whole computation can be done in real-time.

Algorithm 1
Input: The composite function $H(x)$ of degree n;
Output: The function $g(x)$ of degree k that satisfies $f(g(x)) = H(x)$;
DecompositionG($H(x)$)
begin
  for $i := 1$ to $k-1$ **do**
    $A[m,i] \Leftarrow a_{km-i}$;
  end
  for $l := 1$ to m **do**
    $A[l,0] \Leftarrow 1$;
  end
  for $i := 1$ to $k-1$ **do** // Eqn. (3)
    $c_{k-i} \Leftarrow \frac{1}{m}\left(a_{km-i} - \sum_{l=1}^{m-1}\sum_{j=1}^{i-1} c_{k-i+j} A[l,j]\right)$;
    for $l := m-1$ downto 1 **do**
      $A[l,i] \Leftarrow A[l+1,i] - \sum_{j=0}^{i-1} c_{k-i+j} A[l,j]$;
    end
end
$c_0 = 0$;
Construct a polynomial function $g(x)$ using c_i, $i = 0,1,\cdots,k$;
return $g(x)$;

Algorithm 2
Input: The composite function $H(x)$ of degree n;
The function $g(x)$ of degree k.
Output: The function $f(x)$ of degree m that satisfies $f(g(x)) = H(x)$;
DecompositionH($H(x)$, $g(x)$)
begin
  //Assume $g(x) = x^k + c_{k-1}x^{k-1} + \cdots + c_s x^s$
  for $i := 1$ to $k-1$ **do**
    $B[1,i] \Leftarrow c_i$;
  end
  for $l := 2$ to m **do**
    for $i := 1$ to kl **do**
      $B[l,i] \Leftarrow \sum_{j=\max(1,i-k(l-1))}^{\min(k,i-l+1)} c_j B[l-1,i-j]$;
    end
  end
for $l := 1$ to $m-1$ **do** // Eqn. (4)
  $b_l \Leftarrow \frac{1}{c_s^l}\left(a_{sl} - \sum_{p=1}^{l-1} b_p B[p,sl]\right)$;
end
$b_0 \Leftarrow a_0$;
Construct a polynomial function $f(x)$ using b_l, $l = 0,1,\cdots,m-1$;
return $f(x)$;

3. CURVE IDENTITY ALGORITHM

A single curve may undergo a sequence of degree raising, knot refinement, and composition operations. Unfortunately, these operations are non-commutative. That is, the result of a degree raising operation followed by a composition with a function $t(r)$ is different, in general, from the result when the two operations are applied in reverse order. On the other hand, a knot that introduces a potential discontinuity will remain a potential discontinuity even after degree raising or composition operations. However, a single such knot may be manifested as several such knots after these operations since in the higher degree representation of the shape, multiple knots are necessary to preserve the same potential discontinuities.

Given a curve $C(t)$, we apply the knot removal test once and remove all knots whose removal will not affect the shape. Once completed, there are no knots that introduce potential discontinuities and no such knot will emerge later in the curve identification process.

A curve could be degree raised from a quadratic to a cubic, an operation that cannot be represented as a composition. Similarly, a composition with a non-linear function $t(r)$ that changes the speed of $C(t)$ cannot be represented as a degree raising. Therefore, in the second stage of our algorithm, we interleavingly apply degree reductions and/or decompositions, until both attempts fail. We are then left with a curve with no potentially discontinuous knots, that is not a result of a composition, and whose degree is irreducible. We call such a curve an *irreducible curve*.

Let $C_1(t)$ and $C_2(r)$ be two irreducible scalar polynomial curves. Clearly, if the degrees of the two curves are different, the curves are different. Even if the two curves have the same degree, one cannot tell their identity by a simple comparison of their coefficients. The two curves may represent exactly the same polynomial, while spanning different domains, e.g. domains that can partially overlap or, alternatively, domains which are completely disjoint. In Section 3.1, we explain how to bring the two curves to a canonical domain and how to

detect their shared domain, if any, so that their coefficients or control points can finally be compared and their identity can be detected.

Finally, consider two irreducible piecewise-polynomial curves, $C_1(t)$ and $C_2(r)$. Here, one must compare adjacent polynomial segments one after the other between the two curves, determining the overlapping, if any, between the two curves.

3.1 Shared Domain Determination

In this section, we present an algorithm that determines the shared domain of two scalar irreducible polynomial curves of the same degree that are assumed to be similar. This algorithm can be extended to support multivariate and vector-valued irreducible polynomial manifolds, which will briefly be described at the end of this section.

Let $C_1(t)$ and $C_2(r)$ be two irreducible scalar polynomial curves of the same degree k:

$C_1(t) = \sum_{i=0}^{k} a_i t^i, \quad 0 \le t \le 1;$

$C_2(r) = \sum_{i=0}^{k} b_i r^i, \quad 0 \le r \le 1.$

The two curves are assumed to represent exactly the same polynomial, possibly defined on different or completely disjoint domains. There are three main steps in the domain determination algorithm. In the first step, and because of our similar-polynomial assumption, there exists a linear composition transformation $l(t) = at + b$ of the parametric domain such that

$C_1(t) = C_2(at + b), \quad 0 \le t \le 1, a \ne 0.$

Note that the range of $l(t)$ is not necessarily $[0,1]$. Then, using this equality, we can derive the following equation:

$$\sum_{i=0}^{k} a_i t^i = \sum_{i=0}^{k} b_i (at+b)^i = \sum_{i=0}^{k} \left(\sum_{j=0}^{i} \binom{i}{j} a^j b^{i-j} t^j \right) = b_0 \left(\binom{0}{0} a^0 b^0 t^0 \right) + b_1 \left(\binom{1}{0} a^0 b^1 t^0 + \binom{1}{1} a^1 b^0 t^1 \right) +$$

$$+ b_2 \left(\binom{2}{0} a^0 b^2 t^0 + \binom{2}{1} a^1 b^1 t^1 + \binom{2}{2} a^2 b^0 t^2 \right) + \cdots$$

$$\cdots + b_k \left(\binom{k}{0} a^0 b^k t^0 + \binom{k}{1} a^1 b^{k-1} t^1 + \binom{k}{2} a^2 b^{k-2} t^2 + \cdots + \binom{k}{k} a^k b^0 t^k \right)$$

and consequently, we have

$$\sum_{i=0}^{k} a_i t^i = \sum_{i=0}^{k} \left(\sum_{j=i}^{k} b_j \binom{j}{i} a^i b^{j-i} \right) t^i.$$

Now, by comparing the coefficients of t^k and t^{k-1} we get

$$a = \sqrt[k]{\frac{a_k}{b_k}}, \quad b = \frac{a_{k-1} - b_{k-1} a^{k-1}}{k b_k a^{k-1}}.$$

In the second stage, one can evaluate the composition of $C_2(l(t))$ and compare the coefficients of this curve with those of $C_1(t)$, for the curve identity. If indeed all coefficients are found identical, we proceed to the third stage of the algorithm. Otherwise, we declare these two curves as different.

The third step of the algorithm determines whether the parametric domains of the two curves are completely disjoint or partially overlap. In the case of partial overlap, we need to find the shared domain of both curves. The answer is derived by computing the intersection $S_1 \cap S_2$, where $S_1 = [0,1]$ and

$$S_2 = \begin{cases} [a \cdot 0 + b, a \cdot 1 + b] = [b, a+b], & \text{if } a > 0 \\ [a \cdot 1 + b, a \cdot 0 + b] = [a+b, b], & \text{if } a < 0 \end{cases}$$

Then, by mapping the end points of $S_1 \cap S_2$ back into the parametric domain $0 \le r \le 1$ of curve $C_2(r)$, we complete the algorithm.

The presented algorithm can be extended to support vector-valued manifolds. Consider two vector curves $C_1(t)$ and $C_2(r)$ that are assumed to be identical. By applying the presented algorithm (for scalar curves) to each coordinate of $C_1(t)$ and $C_2(r)$, we derive a and b, map and then compare the control points of $C_1(t)$ and $C_2(l(t))$, for identity.

4. EXAMPLES AND APPLICATIONS

We present several examples, starting with examples for the identity tests between two piecewise polynomial parametric curves in Section 4.1. In Section 4.2, we consider the case of watermarking freeform spline geometry, while in Section 4.3, we present an application to matching curves across boundaries of surfaces sharing a common seam.

4.1 Are two curves the same?

We start our examples with an identity comparison between two polynomial curves, CRV1_0 and CRV2_0, which undergo a sequence of degree raising, knot refinement and composition operations, as described in Tab. 1 to 3. The curves are shown in Fig. 1 to 3. The two final curves in each example, for example CRV1_4 and CRV2_3 in Tab. 1, are processed by the proposed algorithm and are either found identical with an overlapping domain that is specified or are found non overlapping and a mismatch is declared. The original data sets are presented in Appendix A.

Curve 1	Curve 2
CRV1_0: Original cubic polynomial	CRV2_0: Original cubic polynomial
CRV1_1: Extract Region [0.1, 0.9]	CRV2_1: Extract Region [0.3, 0.8]
CRV1_2: Refinement at 0.2, 0.33, 0.7	CRV2_2: Refinement at 0.35, 0.4
CRV1_3: Composition with quadratic curve CRV0	CRV2_3: Degree Raise to a quartic
CRV1_4: Degree raise to degree 7th polynomial	
CRV_MATCH: Extract Region [0.4874, 0.8795] of CRV1_4 to match CRV2_3	

Tab. 1. Example 1 - the stages two identical curves undergo; see also Fig. 1. The original data of the curves can be found in Appendix A.

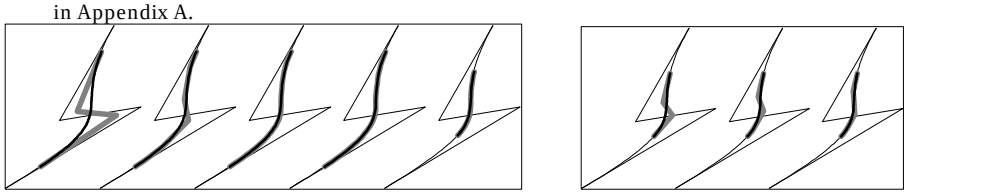

Fig. 1. Example 1 - The different representation changing operations that the two input curves undergo before a similarity test is applied to them. The original identical curves, CRV1_0 and CRV2_0, are shown as thin lines behind the drawn curves (thick black) and their control polygons (thick gray). See also Tab. 1.

Curve 1	Curve 2
CRV1_0: Original quadratic polynomial	CRV2_0: Original quadratic polynomial
CRV1_1: Composition with quadratic curve CRV0	CRV2_1: Extract Region [0.5, 1.0]
CRV1_2: Composition with quadratic curve CRV0	
CRV1_3: Composition with quadratic curve CRV0	
CRV1_4: Degree raise to degree 17th polynomial	
CRV1_5: Refinement at 0.3, 0.5, 0.7	
CRV1_6: Extract Region [0.0, 0.7]	
CRV_MISMATCH: No overlapping region - curves are disjoint	

Tab. 2. Example 2 - the stages two identical curves undergo until non overlapping domain results; see also Fig. 2. The original data of the curves can be found in Appendix A.

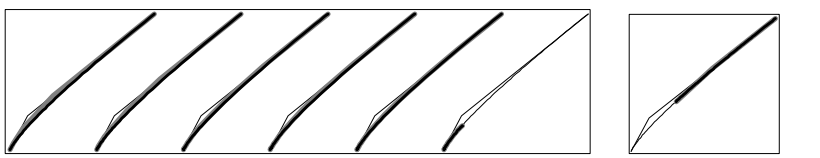

Fig. 2. Example 2 - The different representation changing operations that the two input curves undergo before a similarity test is applied to them. The original identical curves, CRV1_0 and CRV2_0, are shown as thin lines behind the drawn curves (thick black) and their control polygons (thick gray). See also Tab. 2.

Curve 1	Curve 2
CRV1_0: Original cubic polynomial	CRV2_0: Original cubic polynomial
CRV1_1: Composition with cubic curve CRV0_1	CRV2_1: Composition with cubic curve CRV0_2
CRV1_2: Degree Raise twice to 12th degree polynomial	CRV2_2: Degree Raise to 11th degree polynomial
CRV1_3: Refinement at 0.2, 0.3, 0.5, 0.6	CRV2_3: Refinement at 0.2, 0.4. 0.6, 0.9
CRV1_4: Extract Region [0.1, 0.7]	
CRV_MATCH: Extract Region [0.0215, 0.6590] of CRV2_3 to match CRV1_4	

Tab. 3. Example 3 - the stages two identical curves undergo; see also Fig. 3. The original data of the curves can be found in Appendix A.

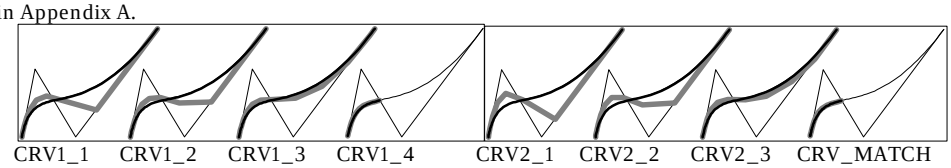

Fig. 3. Example 3 - The different representation changing operations that the two input curves undergo before a similarity test is applied to them. The original identical curves, CRV1_0 and CRV2_0, are shown as thin lines behind the drawn curves (thick black) and their control polygons (thick gray). See also Tab. 3.

4.2 Reparameterization as an Insecure Watermarking

Digital watermarking is a relatively new subject of research in computer graphics. In particular, watermarking three-dimensional data has been mainly focused on polygonal meshes. Watermarking freeform spline geometry has also been the subject of great interest; however, there has been little success in the area so far. As presented in this paper, the same freeform geometry can be represented in different forms when we deal with Bézier and B-spline curves and surfaces.

Not surprisingly, all these techniques for modifying representation were once proposed as possible ways of watermarking freeform geometry represented as B-spline [8] manifolds. Yet, degree raised curves could easily be degree reduced [9]. Similarly, a refined curve could be restored via a knot removal procedure [7]. Ohbuchi et al. [8] mentioned these deficiencies in the watermarking methods based on knot insertion and degree raising and they proposed reparameterization via composition as possibly a more robust alternative to watermarking. Hence, a clear and immediate implication of our result in this paper is the discrediting of composition as a watermarking option.

4.3 Curve Matching via a Reduction

Modern geometric CAD systems are capable of modeling highly complex scenes, with hundreds if not thousands of (trimmed) surfaces. Different surfaces are frequently stitched together along shared seams. In many cases, the curves along the stitched surfaces share neither the same degree nor the same knot sequence. This incompatibility can produce black holes while the two adjacent surfaces are tessellated into polygons (see Fig. 4(c). and Fig. 5(c).). Similarly, there could be discontinuities in parametric texture that are mapped across shared curved boundaries, due to compositions (see Fig. 4(a). and Fig. 5(a).).

Given two curves $C_1(t)$ and $C_2(r)$ from two surfaces along a shared boundary, we consider the problem of checking whether these two curves are identical, and if so, find a canonical representation for the two curves. Such a canonical representation is very useful in matching the tessellation on both sides of the shared seam (see Fig. 4(d). and Fig. 5(d).). Moreover, it may provide a way to reparametrize one of the surfaces so that the two surfaces are texture-mapped seamlessly across their common boundary curve (see Fig. 4(b). and Fig. 5(b).).

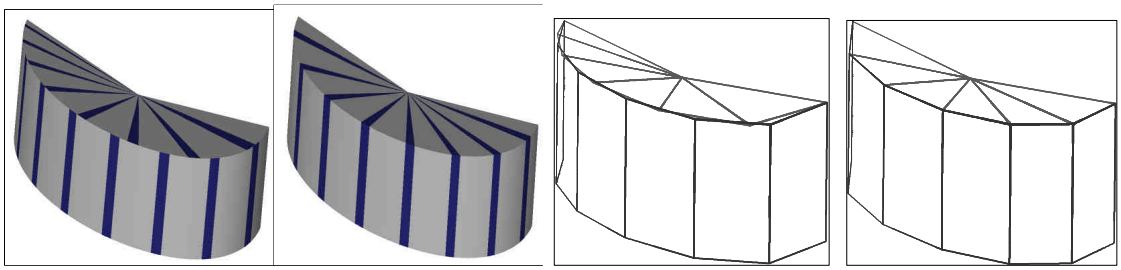

Fig. 4(a). Fig. 4(b). Fig. 4(c). Fig. 4(d).

Fig. 4(a)., Fig. 4(b). The same boundary curve has different representation; and Fig. 4(b)., Fig. 4(d). its canonical form is derived via decomposition, yielding proper texture mapping Fig. 4(c). and tessellation Fig. 4(d). across the seam.

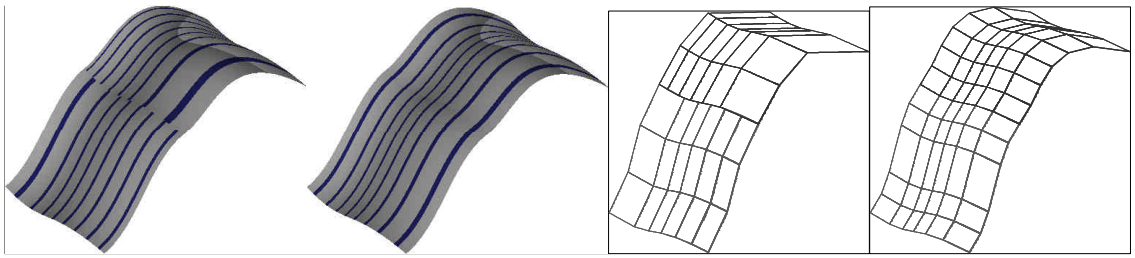

Fig. 5(a). Fig. 5(b). Fig. 5(c). Fig. 5(d).

Fig. 5(a)., Fig. 5(c). The same boundary curve can be parameterized differently. Two differently parameterized boundary curves are matched into one canonical form, yielding seamless texture Fig. 5(b). and tessellation Fig. 5(d).

Fig. 6(a). Fig. 6(b).

Fig. 6(a). A differently parameterized boundary curve (around the neck of a duck model) results in a skewed texture map;
Fig. 6(b). but the matched curve using a canonical form unifies the two surface patches.

5. DISCUSSION AND CONCLUSION

In this paper, we have presented an answer to the fundamental question about the identity of piecewise-polynomial parametric curves. In order to answer this query one is required to have the ability to reverse all possible ways of modifying a curve without changing its trace, namely degree raising, knot refinement and/or composition. Based on these functionalities, we can convert the two curves into their irreducible canonical forms by interchangingly applying these reverse operations. A final stage of shared domain determination is required before the coefficients or the control points can be compared.

The presented results could immediately be applied to piecewise rational forms. Nevertheless, rational forms have more degrees of freedom to change the representation while preserving the shape. Beyond the obvious ability to uniformly scale all weights, the Moebious transformation [6] over rationals provides a simple scheme to modify the curve speed (but neither the trace nor the degree) by changing the weights. A canonical representation that completely fixes this degree of freedom could require that the value of the first weight equal the last.

We believe that the extension of the presented approach to tensor product surfaces is straight forward, as the bivariate function is multiplicative and one can treat each variable separately. This extension also holds for multivariate tensor product representations.
We believe that the similarity test presented in this paper will find other important applications in freeform shape design.

ACKNOWLEDGMENTS
All the algorithms and figures presented in this paper were implemented and created using the IRIT solid modeling system [4] developed at the Technion, Israel. This work was supported in part by the Israeli Ministry of Science Grant No. 01–01–01509, in part by the Israel Science Foundation (grant No. 857/04), and in part by the European FP6 NoE grant 506766 (AIM@SHAPE), in part by the Korean Ministry of Information and Communication (MIC) under the Program of IT Research Center on CGVR, in part by the Korean Ministry of Science and Technology (MOST) under the Korean-Israeli binational research grant, and in part by grant No. R01-2002-000-00512-0 from the Basic Research Program of the Korea Science and Engineering Foundation (KOSEF).

APPENDIX A

A.1 Example 1

CRV0: A scalar quadratic Bezier curve with coefficients $(0.0, 1/8, 1.0)$.

CRV1_0 = CRV2_0: A cubic Bezier curve with control points $(0.0, 0.0), (10.0, 6.0), (4.0, 5.0), (8.0, 12.0)$.

A.2 Example 2

CRV0: A scalar quadratic power basis curve with coefficients $(0.0, 0.4, 0.6)$. That is $0.4t + 0.6t^2$.

CRV1_0 = CRV2_0: A quadratic Bezier curve with control points $(0.0, 0.0), (1/8, 1/4), (1.0, 1.0)$.

A.3 Example 3

CRV0_1: A scalar cubic power basis curve with coefficients $(0.0, 0.0, 1/2, 1/2)$. That is $t^2/2 + t^3/2$.

CRV0_2: A scalar cubic power basis curve with coefficients $(0.0, 1/4, 1/4, 1/2)$. That is $t/4 + t^2/4 + t^3/2$.

CRV1_0 = CRV2_0: A cubic Bezier curve with control points $(0.0, 0.0), (1.0, 5.0), (4.0, 0.0), (10.0, 8.0)$.

REFERENCES

[1] Cohen, E., Riesenfeld, R. F., and Elber, G., *Geometric Modeling with Splines: An Introduction*, A. K. Peters, Natick, MA, 2001.
[2] Gathen, J. von zur, Functional decomposition of polynomials. *J. Symb. Comput.*, Vol. 9, 1990, pp 281-299.
[3] Heal, K. M., Hansen, M., and Richard, K., *Maple V Learning Guide (for Release 5)*, Springer-Verlag, 1997.
[4] *IRIT 9.0 User's Manual, October 2002*, Technion. http://www.cs.technion.ac.il/~irit.
[5] Kozen, D. and Landau, S., Polynomial decomposition algorithms, *J. Symb. Comput.*, Vol. 7, 1989, pp 445-456.
[6] Lee, E. and Lucian, M. Moebius Reparametrization of Rational B-splines. *Computer Aided Geometric Design*, Vol. 8, 1991, pp. 213-238.
[7] Lyche, T., and Morken, V., Knot removal for parametric B-spline curves and surfaces, *Computer Aided Geometric Design*, Vol. 4, No. 3, 1987, pp 217-230.
[8] Ohbuchi, R., Masuda, H., and Aono, M., A shape-preserving data embedding algorithm for NURBS curves and surfaces, *Proc. of Computer Graphics International (CGI 99)*, Canada, June 1999.
[9] Piegl, L., and Tiller, W., Algorithm for degree reduction of B-spline curves, *Computer-Aided Design*, Vol. 27, No. 2, 1995, pp 101-110.
[10] Spiegel, M., *Mathematical Handbook of Formulas and Tables*, Schaum's Outline Series in Mathematics, McGraw-Hill, pp 4, 1968.
[11] Wolfram, S., *The Mathematica Book*, 3rd ed., Wolfram Media/Cambridge University Press, 1996.