펜-입력 디스플레이에서의 큐빅 B-Spline의 스케치-기반 편집 방법 연구

(Sketch-based Modification of the Cubic B-Splines for the Pen-input Displays)

김 대 현 † 김 명 준 ††
(Dae-Hyun Kim) (Myoung-Jun Kim)

요 약 LCD 타블렛과 타블렛 PC 등과 같은 펜-입력 디스플레이는 CAD에서, 특히, 컨셉트 디자인 단계에서 유용하게 사용될 수 있다. 본 논문에서는, 이런 장치들을 대상으로 하는 CAD 시스템에서 사용될 수 있는 직관적인 B-Spline 수정 방법을 제안한다. 컨트롤 포인트를 기반으로 하는 B-Spline 수정 방법과는 달리, 본 논문에서 제안한 방법은 곡선이나 곡면의 최종적인 모양을 결정하는데 사용자의 펜 입력을 이용한다. 이전에 방법에서 컨트롤 포인트나 조작자(manipulator)를 이용했던 것에 비해 원하는 모양을 만들어 가는 데 소요되는 시간을 줄여준다. 이런 과정은 디자인을 하는데 있어서 지금까지는 피할 수 없는 것처럼 여겨져 왔다. 본 논문에서 제시된 방법의 용이성을 검증하기 위하여 여러 명의 디자이너들과 함께 실험을 수행한 결과를 보여준다.

키워드 :

Abstract Pen-input displays, such as LCD tablets and tablet PCs, are already popular for its usability in CAD design, in particular, in concept design phase. We propose an intuitive *B-Spline* modification scheme that can be used for the CAD systems targeting for such devices. Differently from the control point based modification schemes for the *B-splines*, our scheme relates user pen marking to determining the final shape of the target curve and surface. This, eventually, reduces time for interacting with the shape parameters (i.e., by control points or direct manipulators), which has been regarded as an unavoidable routine tasks for design. To prove its usability, we made an experiments with selected subjects who have been working for industrial design.

Key words : Pen-input displays, B-Splines, Concept Design, CAD

1. Introduction

A significant achievement for one-handed free-form drawings was made by Baudel [1]. He proposed a brand new method by which users can freely modify their drawings with successive pen markings; for example, the user gives a pen marking over an existing curve $C(u)$, and the system sculpts the curve considering the shape of the pen marking--for a perceptive overview, see Fig. 1. One restriction of this approach is that it uses *off-line spline curves*, which means that the edited curves are represented by *piecewise linear curves*; the program generates G^1 continuous Bezier curves only after an explicit user request (e.g. by pushing a button). Adobe Illustrator uses this scheme but is limited to *Bezier curves* [2].

There exist different curve drawing methods by which the user can use both hands in drawing. Singh proposed an interactive method to convert a B-Spline curve into stylish French curve segments by iteratively breaking the curve, fitting them with elliptic curves, and merging them together [3]. Balakrishnan et al. implemented tape drawing method which has been used to draw smooth curves in car design [4].

† 정 회 원 : (재)그래픽스연구원 Tiled Display Team 연구원
daek@acm.org
†† 정 회 원 : 이화여자대학교 디지탈미디어학부 교수
mjkim@ewha.ac.kr
(Corresponding author임)
논문접수 : 2005년 5월 24일
심사완료 : 2005년 11월 29일

Sachs et al. proposed 3D curve sketching environment, 3-Draw, for a virtual environment [5]. In 3-Draw, the user holds a stylus in one hand and a tablet in the other. These tools serve to draw 3D curves but they were not concerned about the representation of the drawn curve and intuitive modification of the drawn curve. Krueger suggested similar environment using computer vision techniques to manipulate four control points along a spline curve [6]. Poston et al. and Cutler et al. also suggested two handed input for virtual environment, however, they are more concerned about fluent user interactions rather than inventing a curve drawing method [7,8].

We now focus on the B-spline curve editing. Fowler et al. [9] found a new way to directly manipulate B-Spline curves, using constrained minimization techniques. This method is inherently dependent on the structure of the knot vector. For example, when the knot vector is dense, the locality of the modification becomes small, leaving the burden to have a good knowledge over the knot structures.

Banks et al. [10] proposed a new method to edit B-Spline curves by cutting and sketching control polygons. It imposes an additional burden on the user to understand what control points are, rather than curve shapes. However, their examples revealed that editing the control polygon by a pen marking is quite intuitive, although it is not as intuitive as a direct curve manipulation. At the end of Section 2, we show that the our algorithm generates control polygon of the curves in a similar way as the user sketched a control polygon in [10].

Zheng et al. proposed a new method to deform a curve by matching it with another input curve [11]. The authors used knot removal techniques [12] which increase computation time and sacrifice smooth shape changes in the portion of the transition between the original curve and the input curve.

Recently, Grossman et al. proposed a method to draw nice curves using traditional tape drawing; their achievement is about allowing the user to draw aesthetic curves such as car body line. For this, large display wall is needed and special interaction devices have been invented too [13]. Compared to this approach, ours are more general and targeted for small-scale applications such as Adobe Illustrators and plug-ins for existing 3D modeling systems.

Applying the Baudel's approach, which seems to us most fit for pen-input display, to the B-Spline curves brings about two problems. First, each pen input needs more knots to be inserted into the existing knot vector, in particular, around the transition part between the original curve and the sketched curve. Therefore, iterative sketches on the existing curve will end up with unnecessarily many knots; considering that multiple knots can reduce the continuity of the final curve, it is desirable to have only necessary knots to keep the desired shape. Second, similar to the first problem, often user may want to simplify the existing curve by giving a new pen marking, as shown in Fig. 9. In this case, the user should know about knot structure.

Michaelik et al. [14] suggested a method to sketch B-Spline curves but did not detailed how to address the aforementioned problems. Our approach here combines two methods from Fowler et al. [9] and Baudel [1]; by which it resolves the locality constraints inherent from B-Spline knot structure in the pen-input sketching environment. The former modifies B-Spline curve based on the underlying knot structure of the curve and is very restricted in determining the locality of modification; for example, if there are dense knots for the B-Spline curve, moving a curve point will change only very small region of the curve regardless user's intention. Meanwhile, the latter works only on Bezier curves. Our approach takes the two approaches such that the smooth B-Spline curves can be sketched through pen-input; given a pen marking drawn over a B-Spline curve, as shown in Fig. 1, the algorithm defines how to modify the curve considering both the pen marking and the existing curve, resulting in an implicitly defined shape change. The main contributions of our work can be summarized as follows:

- **Direct control with implicitly defined locality constraints** In this method, the user's pen

markings are directly applied to a cubic B-Spline curve. Locality means that part of the curve for sculpting is implied by the shape of the input pen marking.

- **On-the-fly C^2 curve** The cubic B-Spline curve provides C^2 continuity all over the curve unless it has redundant knots inside. Therefore, the pen marking, which locally and directly operates on the curve, will produce another C^2 continuous B-Spline curve.
- **Fewer knots** The algorithm incrementally adds new knots. Thus only necessary knots are introduced in the transition area between the original curve and the pen marking. For example, when the original curve has a complicated shape (and thus many control points) a simplifying stroke may modify the curve to contain fewer control points.

This paper also demonstrates how this approach can be used in deforming surfaces intuitively as well. First, we show how to re-sketch a parameter line of a B-Spline surface, resulting in a surface deformation. Second, we re-sketch a wire (B-Spline curve) that is bound to a surface, resulting in an implicit surface deformation described in [15].

2. Overview

Let us first assume that we have the following cubic B-Spline curve:

$$C(u)=\sum_{i=0}^{n} B_i P_i$$

where P_i is a control point and B_i is a B-Spline basis function over a knot vector T:

$$T=\{t_0,t_1,t_2,t_3,\cdots,t_{n+1},t_{n+2},t_{n+3},t_{n+4}\}$$

Fig. 1 shows the overall procedure to be taken to modify the given B-Spline curve $C(u)$:

- STEP1: The user gives a pen marking, $h=\{h_i|i=0,\cdots,m\}$.
- STEP2: The algorithm computes the shape change in a digitized point sequence that consists of the following (See also Fig. 1):
  - h: The input pen marking.
  - B^l,B^r: Smooth transitions from h to the existing curve (dotted curve in the middle of Fig. 1).

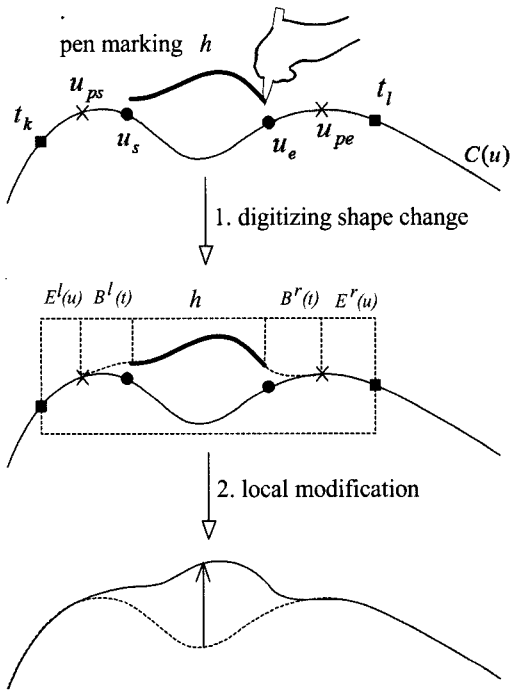

Figure 1 Top: The target B-Spline curve $C(u)$, and the user-drawn modification pen marking h. Middle: Computing the shape change considering both the knot vector of the existing curve and the given pen marking. Bottom: Local modification to the existing curve

  - E^l,E^r: The remaining curve portions within the knot interval which contains each end of the transitions.
- STEP3: The algorithm locally modifies the $C(u)$ according to the digitized shape change.

Here, we assume that the user is iteratively modifying the existing curve but not drawing a new curve, as shown in Fig. 10; when the user wants to draw a new curve, the user has to explicitly change the mode, for example, by pushing a button. Also, since C^0 continuity can be achieved by simply connecting two C^2 curves and simply adding another user interface, this paper focusses only on C^2 continuous sculpting; any user pen marking placed near an existing curve results in a C^2 continuous local modification to the curve.

The remainder of this paper is organized as follows. Section 3 explains how to compute the shape change intended by the user pen marking. Section 4 explains constrained curve modification to the cubic B-Spline curve. Section 5 demonstrates how to use this B-Spline curve modification scheme to the B-Spline surfaces.

3. Computing the Shape Change

This section details STEP 2 in Section 2. To properly extract the curve portions that we will piece together with the input pen marking h to form the shape change, we need to locate several break points that divide the existing curve. In the following, we detect the breaking points in the parameter domain of $C(u)$, $\{t_k, u_{ps}, u_s, u_e, u_{pe}, t_l\}$, as shown on the top of Fig. 1.

We first establish approximate correspondence between $C(u)$ and h; from the parameter domain of $C(u)$, locate an interval $[u_s, u_e]$, which approximately corresponds to the pen marking by simply searching for the closest points in $C(u)$ from two endpoints of the pen marking. As a result, as shown in Fig. 1, $C(u)$ of $u \in [u_s, u_e]$ constitutes the approximate correspondence.

To determine where $B^l(t)(B^r)$ starts or ends in the parameter domain of $C(u)$, we define two terms, $u_{ps} = u_s - d_1$ and $u_{pe} = u_e + d_2$; where $d_1 = 3|C(u_s) - h_0|$ and $d_2 = 3|C(u_e) - h_m|$. For a broader blending area, we could choose a value larger than three.

The two curve portions corresponding to the two intervals, $[u_{ps}, u_s]$ and $[u_e, u_{pe}]$, will be used to make a continuous geometric transition from $C(u)$ to the pen marking h. Now from the knot vector T, look for the t_k and t_l: $t_k = \max\{t_i \le u_{ps} | t_i \in T\}$ and $t_l = \min\{t_i \ge u_{pe} | t_i \in T\}$. We now have all the breaking points we need to decompose the curve $C(u)$. Subsequently, only the left part of the curve will be explained, since the right part functions analogously.

- Call the curve portion $C(u)$ of $u \in [t_k, u_{ps}]$, $E^l(u)$.
- Convert $C(u)$ of $u \in [u_{ps}, u_s]$ into a piecewise linear curve, $B^l(t)$, which has chord-length parametrization, $t \in [u_{ps}, u_s]$
- Blend $B^l(t)$ with a line segment L which is tangent at h_0 and has length d_1, and keep it into $B^l(t)$ itself [1].

We now know how the original curve should be changed by the shape change embodied by several curve portions, $E^l(u)$, $B^l(u)$, h, $B^r(u)$, and $E^r(u)$ (See also Fig. 1). Piece them together into one piecewise linear curve $\bar{h}$ with the parameter domain $[t_k, t_l]$. In the next section, $\bar{h}$ will be approximated with $C_{loc}(u)$, which will replace part of $C(u)$, $u \in [t_k, t_l]$, resulting in the local modification. At a first glance, $E^l(u)$ seems unnecessary. To modify $C(u)$ only within the domain $[u_{ps}, u_{pe}]$, inserting multiple knots at u_{ps} is required. This will introduce unnecessarily many knots; several times of sketching over the $C(u)$ will introduce uncontrollably many knots, which at some point will need running knot removal algorithm. Therefore, we let the B-Spline curve $C_{loc}(u)$ approximate the piecewise linear curve over knot interval $[t_k, t_l]$, iteratively minimizing pointwise errors and adding necessary knots.

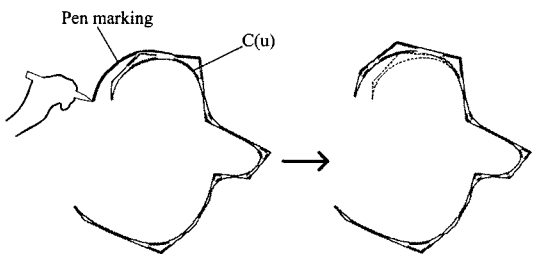

Figure 2 An example shows the shape change exceeds the boundary of the existing curve $C(u)$

Also the case in which $u_s - d_1$ exceeds the parameter boundary of $C(u)$ should be considered, as shown in Fig. 2. There are two possibilities: The first one is to let $B^l(t)$ (or h) replace the left part of $C(u)$. The second one lets the final curve interpolate the end point of $C(u)$. Here we choose the second approach since it lets users easily extend the curve at the end points.

4. Local Modification of the B-Spline Curve $C(u)$

This section explains how to iteratively approximate the shape change, $\bar{h}$, with a cubic B-Spline curve $C_{loc}(u)$ and replace part of $C(u)$ with that; conceptual overview is shown in Fig. 3. Therefore, the final curve is obtained by locally modifying $C(u)$ through a user pen marking. The new curve $C_{loc}(u)$ is constrained to follow the target curve $C(u)$ by sharing the control points (which are called *constrained control points*) near the transition part with those of $C(u)$; for example, two control polygons coincide in part as notified in Fig. 3. The iterative approximation process for $C_{loc}(u)$ starts with a new knot vector with the knot interval $[t_k,t_l]$ empty; the algorithm iteratively adds new knots in $[t_k,t_l]$ and computes the unknown control points using constrained least squares. Changing the knot vector T into T_{loc} and later adding a new knot into T_{loc} causes changes in the constrained control points as well; the algorithm checks whether the constrained control points from $C(u)$ are influenced (i.e., refined) by the knot insertion algorithm [16]. Usually, inserting a new knot around t_{k+1} would refine the control polygon and shorten (or extend) the edge $\overline{P_{k-3}P_{k-2}}$. This influence can be checked by looking at knot span $[t_k,t_{k+2}]$ (and $[t_{l-2},t_l]$) of T; to simplify the description of the algorithm, in the following, we define *mark*.

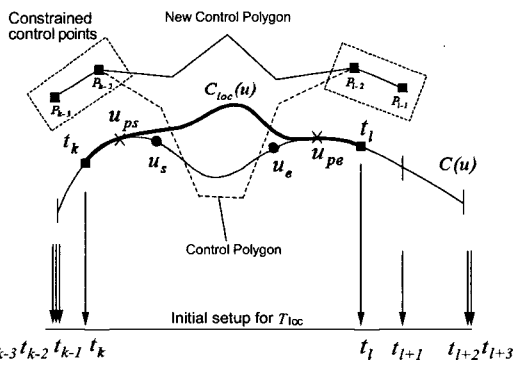

Figure 3 A new knot configuration for the local curve $C_{loc}(u)$and the constraints imposed on both curves

First, we need to initialize a new knot vector T_{loc} for $C_{loc}(u)$.

- Collect knots from the knot vector, T, of $C(u)$ and make a new knot vector for $C_{loc}(u)$,

$$T_{loc}=\{t_{k-3},t_{k-2},t_{k-1},t_k,t_l,t_{l+1},t_{l+2},t_{l+3}\}$$

If there are more than one knot in (t_k,t_l) of T, create two marks to watch any change to the constrained control points:

$$z_k=t_{k+1},z_l=t_{l-1}.$$

Let us note that, constructing the knot vector T_{loc} from T, all the knots in (t_k,t_l) have been ignored since they do not reflect the parametrization of h any longer. Instead, new knots will be inserted while approximating $\bar{h}$ with $C_{loc}(u)$.

The *marks* simplifies the algorithm to keep track of the knots, newly inserted, at the intervals $[t_k,t_{k+2}]$ (and $[t_{l-2},t_l]$), so that we could notice when and how the constrained control points from $C(u)$ are affected.

Overall algorithm can be summarized as the following procedure:

Algorithm 1-1: LocalCurveApproximation

INPUT: $C(u)$, T_{loc}, tolerance ε, and h

CHANGE: $C_{loc}(u)$ with T_{loc} updated

(1) Set up the **constraints** = the constrained control points (P_{k-3}, P_{k-2}) and (P_{l-2}, P_{l-1})

loop

(2) $C_{loc}(u)$ = ConstrainedLeastSquares (**constraints**, T_{loc}, h);

(3) Error = FindMaxError ($C_{loc}(u)$, h);

(4) **if** Error $< \varepsilon$, return CONVERGENCE

(5) i = TraceMostDeviatedKnotInterval($C_{loc}(u)$, h, T_{loc});

(6) SubdivideKnotVector (**constraints**, $\frac{t_i+t_{i+1}}{2}$, T_{loc}, $C_{loc}(u)$);

In step (1), the constraints for the new curve $C_{loc}(u)$ are set. The control points of $C(u)$ that influence the curve segment within $[t_k,t_{k+1}]$ and $[t_{l-1},t_l]$ are $(P_{k-3},P_{k-2},P_{k-1},P_k)$\ on the left side and $(P_{l-4},P_{l-3},P_{l-2},P_{l-1})$ on the right side (See also Fig. 3). For the first iteration, therefore, (P_{k-3},p_{k-2}) and (P_{l-2},P_{l-1}) are used as *constraints* for the *ConstrainedLeastSquares* in step (2) (for more details on the constrained optimization, we refer the readers to [17]). However, in a certain

condition, which may occur from the next iterations inserting a new knot, P_{k-2} and P_{l-2} are to be modified in step (6); we discuss how to set up the constraints later.

In step (3), *FindMaxError* measures the maximum error: suppose the piecewise linear curve $\overline{h}$ has the following chord-length parameterization:

$$\{u_0,\cdots,u_m\},\ u_0=t_k, u_m=t_l.$$

Then the maximum error M is defined as follows:

$$M=\max_{i=0}^{m}|C_{loc}(u_i)-h_i|$$

If the maximum error is less than the input tolerance ε the output curve $C_{loc}(u)$ is considered converged and the algorithm stops. *TraceMostDeviatedKnotInterval* of the step (5) searches for the i-th knot span of T_{loc}, which has the most deviated error, and determines a new knot:

$$t^*=(t_i+t_{i+1})/2.$$

In step (6), *SubdivideKnotVector* inserts the new knot t^* into T_{loc}. The above algorithm takes (P_{k-3},p_{k-2}) for the constraints, which come from $C(u)$. When the new knot is inserted near a mark, (P_{k-3},p_{k-2}) of $C(u)$ are also influenced by a B-Spline knot insertion algorithm [16]; thus $C_{loc}(u)$ takes the constrained control points accordingly modified by the knot insertion algorithm. The marks are used to see when these constrained control points are affected by the knot insertion.

To explain the influence we need some new definitions. Suppose two marks, z_k and z_l are inside the T_{loc}.

Definition 1 (Multiplicity of a mark z: $\mu(z)$)

The number of the knots of the T_{loc} that are located in $(z-\epsilon,z+\epsilon)$, counting in the mark z itself, is called the multiplicity of the mark; denote it by $\mu(z)$.

Definition 2 (Neighborhood of a mark z: $N(z)$)

The neighborhood of the mark is an open interval $(t_{prev},t_{\neq xt})$, where t_{prev} $(t_{\neq xt})$ is the closest knot to z, approaching from the left (right) side of z. Denote the neighborhood of z by $N(z)$.

Whenever a new knot inserted into the neighborhood of a mark, the constraints, (P_{k-3},p_{k-2}) and (P_{l-2},P_{l-1}), should be modified, according to the

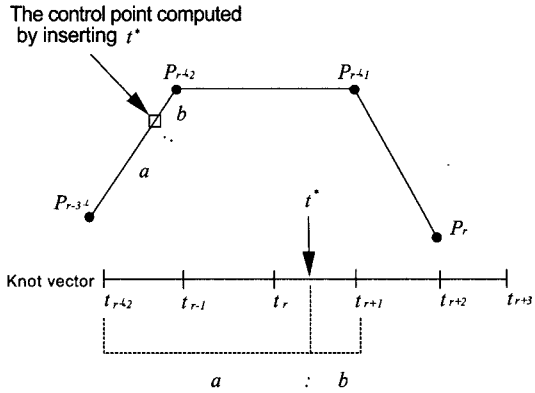

Figure 4 Knot insertion and its effect on the first edge. The control points that affect curve points in the knot span $[t_r,t_{r+1}]$ are shown. Other control points newly generated by inserting t^* are omitted

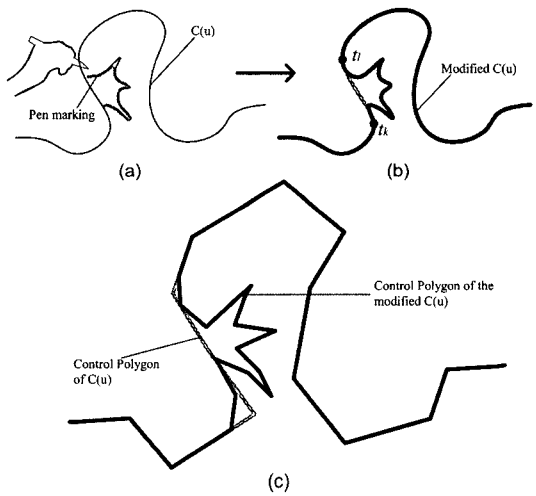

Figure 5 Re-sketch of a B-Spline curve. (a) Shows user's hand giving a pen marking (gray color). (b) Shows the resulting curve locally deformed from the original curve. (c) Shows the control polygons of the previous curve (grey) and the resulting curve (black). In this example, the new curve is more complex than the old one

B-Spline knot insertion algorithm [16]; Fig. 4 illustrates the knot insertion principle. From this principle, for example, when a new knot t^* is inserted into the *neighborhood* of a mark, z_k, the control point P_{k-2} is modified as follows:

$$P_{k-2}=(1-\frac{t^*-t_{k-2}}{z_k-t_{k-2}})P_{k-3}+(\frac{t^*-t_{k-2}}{z_k-t_{k-2}})P_{k-2} \quad (1)$$

Let us note that when t^* goes beyond the z_k to the right the resulting P_{k-2} is an extrapolation point of the line segment $\overline{P_{k-3}P_{k-2}}$; otherwise, it is an interpolation point. In the same way as the above equation, when the new knot is inserted into the neighborhood of z_l, the control point P_{l-2} is modified as follows:

$$P_{l-2}=(1-\frac{t^*-z_l}{t_{l+2}-z_l})P_{l-2}+(\frac{t^*-z_l}{t_{l+2}-z_l})P_{l-1} \quad (2)$$

We now define the step (6), *SubdivideKnotVector*, using the above definitions:

Algorithm 1-2: SubdivideKnotVector

INPUT: **constraints**, t^*, T_{loc}, and $C_{loc}(u)$

CHANGE: T_{loc} and the **constraints** updated

if no marks are defined

insert t^* into T_{loc}

$z_k = z_l = t^*$

Update the **constraints** according to Eq.1 and Eq.2

else

if $\mu(z_k) \geqq 1$ and $t^* \in \mathbf{N}(z_k)$

Update the **constraints** according to Eq.1

$z_k = t^*$, insert t^* into T_{loc}

if $\mu(z_l) \geqq 1$ and $t^* \in \mathbf{N}(z_l)$

Update the **constraints** according to Eq.2

$z_l = t^*$, insert t^* into T_{loc}

This approach using interpolation and extrapolation of control points in accord with the newly inserted knots enhances the quality of local modification because it properly sets up the constraints. Therefore, it can internally manipulate control polygon like Banks' software [10] that lets the user manually sculpt the control polygon itself; such sculpting effect is well shown at the bottom of Fig. 6, 7 and 8. An example of extrapolating control points can be found in Fig. 8 where the user intends to simplify the target curve.

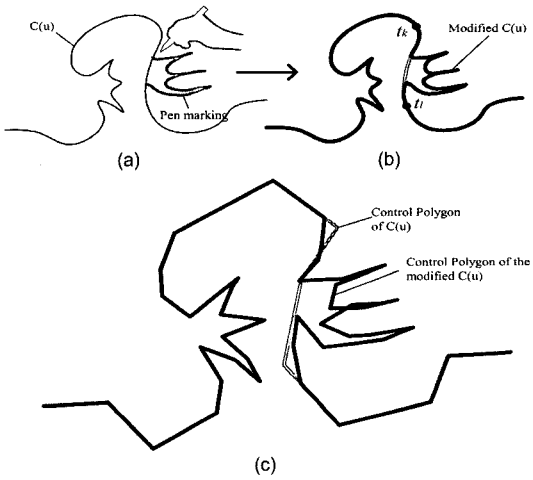

Figure 6. Shows a situation similar to Fig. 5

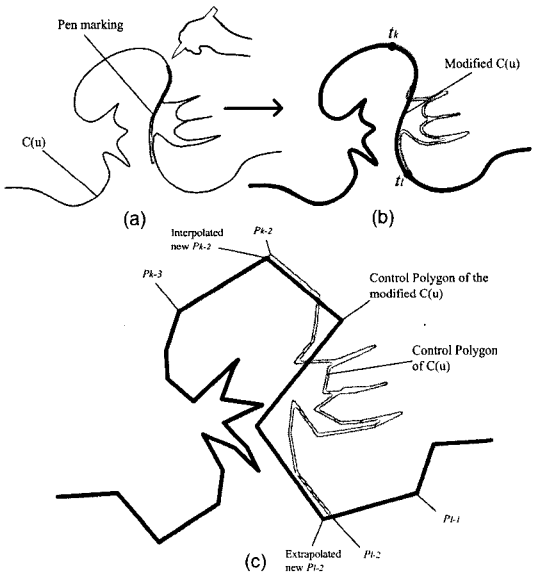

Figure 7. Shows a radical simplication on the B-Spline curve from the result of Fig. 6 by a pen marking. Note that the control polygon of the resulting curve is, in part, an extension of an edge of the control polygon of the old curve (extrapolation) and, in part, the result of cutting an edge (interpolation)

5. Results and Applications

Now we discuss the implementation issues of our algorithm and also present more results in 2D curve sketching as well as 3D curve sketching that leads to surface sketching. We also made a user

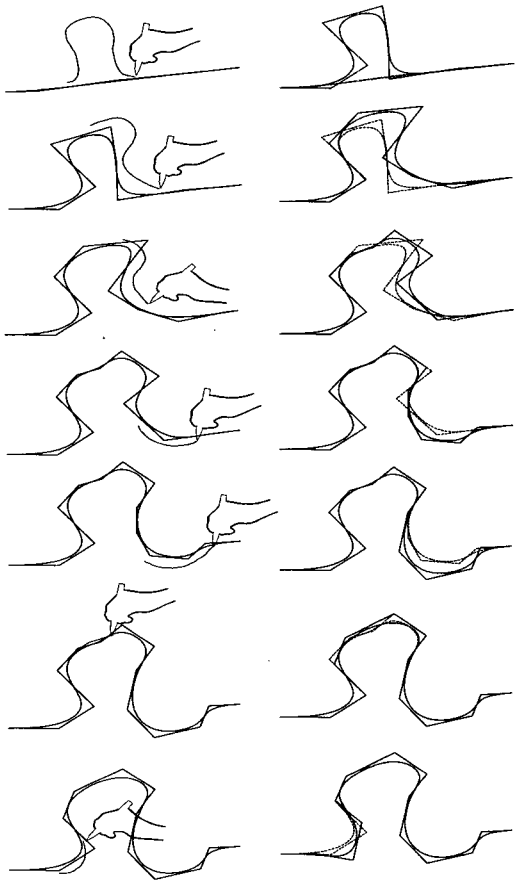

Figure 8 Left Column: The user successively sketches the given B-Spline curve, i.e., almost straight line, towards a shape. Right Column: The algorithm modifies the curve (from dotted to solid curve) according to the sketched pen markings

test where the selected users (experienced designers) draw curves using our curve drawing tool and a commercial modeling S/W MAYA [18].

5.1 Implementation

We implemented the sketching algorithm described using the Microsoft Visual C programming language and tested on a Fujitsu 1.4 GHz Tablet PC running under the Windows XP Tablet PC Edition. We chose OpenGL as a graphics library and used a public domain library to represent and manipulate B-Spline curves and Surfaces. The algorithm runs interactively for 2D curve sketching, but shows a certain response time to apply this to 3D surface editing; this latency does come from deforming 3D surface.

Fig. 8 shows that the user successively sketches the existing curve (left column) and the algorithm locally modifies the B-spline curve. The remainder of the figures demonstrates the B-Spline surface sketching applying the curve sketching metaphor.

5.2 B-Spline Surface Sketching

We now show how to incrementally sketch B-Spline surfaces using the algorithm developed in Section 3 and Section 4. In this section two methods to sketch B-Spline surfaces, applying the curve sketching method, will be demonstrated:

- Surface sketching by re-sketching a u-curve (or a v-curve) of the existing surface.
- Surface sketching by re-sketching an arbitrary B-Spline curve bound to the existing surface, which has been proposed in [15].

For the first method, a pen marking is used to extract a u-curve (or a v-curve) from the target surface; in the same way as the previous sections, the user re-sketches the curve, but in 3D, and the system deforms the surface accordingly.

On a *bicubic B-Spline surface*, as an initial interaction the user draws a pen marking, which is projected onto the uv parameter domain of the surface. Least squares fitting is applied to determine the closest u-curve (or v-curve, subsequently v-curve) to the marking (See Fig. 9 top left). A further user interaction comes to setup an auxiliary surface; auxiliary surface is, in this case, defined to be a ruled surface that passes though the u-curve along a user defined direction. Then the user re-sketches the u-curve to deform the surface(See Fig. 9 top right). The re-sketched pen marking will be projected onto the auxiliary surface; the locally modified resulting curve is a 3D curve since the auxiliary surface is not necessarily to be a plane. To deform the surface as well, we need to solve a system of linear equations as defined in [9].

The step (5) of Alg.1-1, knot selection, needs a slight modification to consider the surface knot space. Newly added knots for the u-curve re-sketched would increase the number of control points of the target surface after each sketch interaction, and the number of knots will eventually

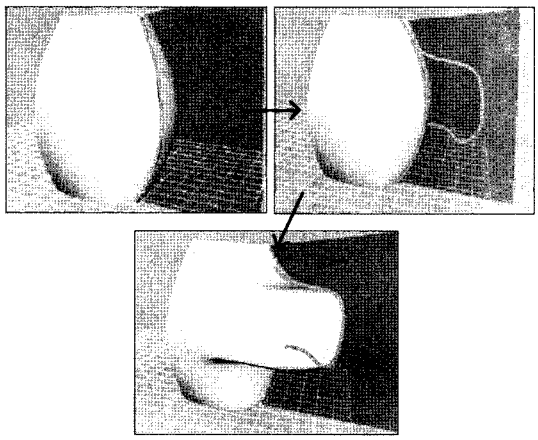

Figure 9 Sketching B-spline surfaces. Top Left: A default auxiliary surface, which is developable (red transparent surface), is constructed perpendicular to the u-curve (or the v-curve). Top right: The user draws a curve onto the auxiliary surface to sculpt the actual surface. Bottom: The resulting surface is shown

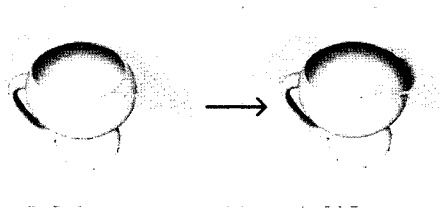

Figure 10 Sketching a jug model. Left: A default auxiliary surface on the boundary curve. Right: the user draws a marking onto the auxiliary surface to make a mouth of the jug

explode after several steps of deformation.

Therefore, a knot vector of the surface before the deformation is reused for the knot selection. Two knot vectors from the u-curve and from the surface must be compatible with each other. As noted in Section 4, when the pen marking simplifies the parameter line, a knot removal algorithm [12] can be used to simplify the knot vector of the target surface as well. Another example of surface deformation by re-sketching a parameter line of a B-Spline surface is demonstrated in Fig. 10.

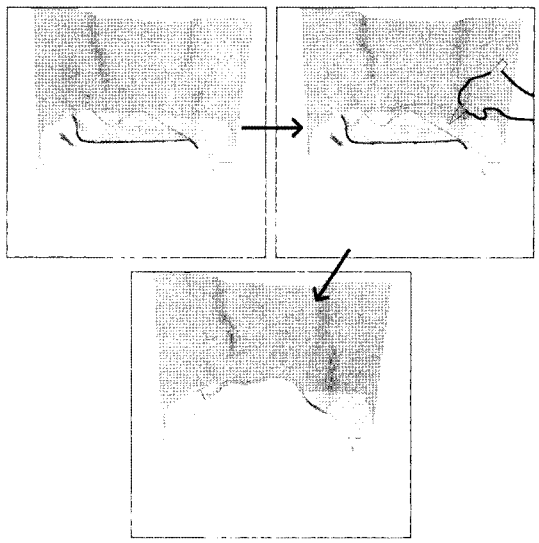

Figure 11 A sketching example for more complex geometry

The second method, re-sketching an arbitrary curve on a target B-Spline surface and deforming the surface to exactly follow the surface curve involves a singular matrix to be solved [14]; moreover, this singular system is mostly ill-conditioned and causes unexpected results. Such an exact method, however, is not necessary for interactive sketching applications. Therefore, to demonstrate more complex examples of sketching B-Spline surfaces, we apply our curve sketching to the WIRES of [15]. In this method, deforming the curve bound to the target surface leads to implicitly deforming the surface as well. In Fig. 11, an intersection curve between the target surface and the auxiliary surface (in this case, the auxiliary surface is an arbitrary surface) is re-sketched and the target surface is deformed following the re-sketched B-Spline curve projected onto the auxiliary surface.

5.3 User Tests

For more quantitative evaluation, we conducted a user test where we selected ten subjects to draw a car silhouette using both our system and a commercial 3D modeling tool MAYA [18]. In this test, the observer watched each subject drawing the target curves on a car image embedded into each system as shown in Fig. 12. We measured

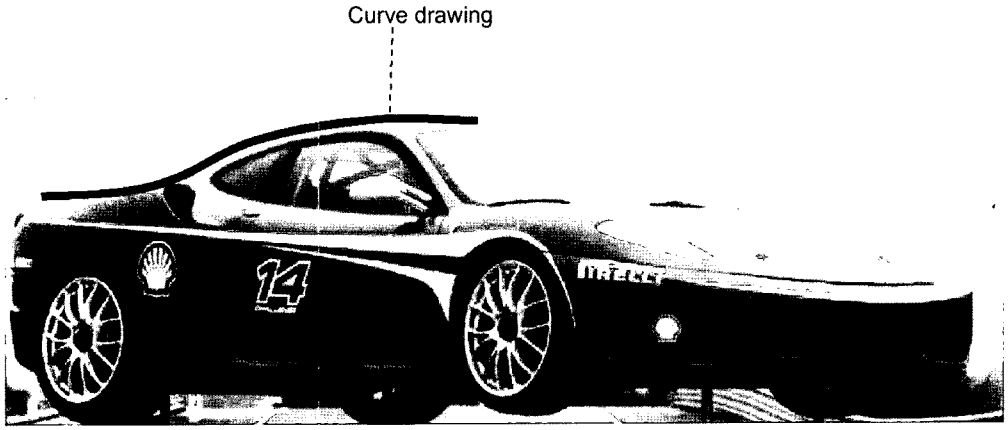

Figure 12 Drawing curves on a car body

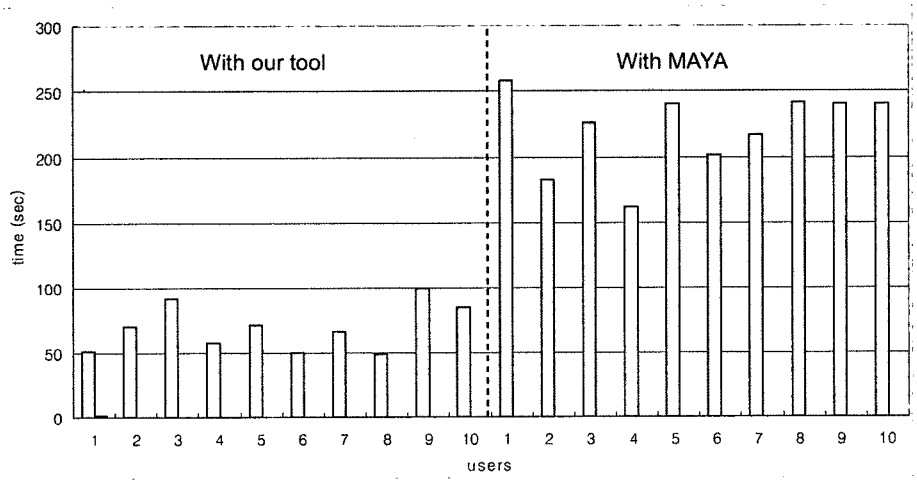

Figure 13 Test results: left part shows times taken to draw the curves with our tool for each subject. Right part shows times with the commercial tool

the time for completing the task.

Fig. 13 shows the statistics for the two tools; our method shows better performance. The main reason behind this result comes from the fact that the commercial tool requires modifying control points whenever the designers have to change the shape. Moreover, they have to change the knot vector when the figure requires more curvy shape. Meanwhile, with our system, they do not need to care about those shape parameters.

6. Conclusion and Future Work

This paper proposed and implemented the constrained cubic B-spline curve modification based on pen-input, which has been proven to be a useful tool for sketching smooth curves and surfaces. As a future work, we suggest to bring this into a virtual 3D environment where the user can draw 3D curve more freely with 3D pointing devices; which has been otherwise quite awkward for the users using pen-input display (i.e., by projecting 2D sketching onto an auxiliary surface), as demonstrated so in Section 5.

References

[1] T. Baudel, A mark-based interaction paradigm for free-hand drawing. In UIST 94, pages 185-192. ACM SIGGRAPH, 1994.
[2] Adobe. Adobe Illustrator 10, 2004.
[3] Karan Singh. Interactive curve design using

digital french curves. In Symposium on Interactive 3D Graphics, pages 20-30. ACM, 1999.
[4] Ravin Balakrishnan, George Fitzmaurice, Gordon Kurtenbach, and Willian Buxton. Digital tape drawing. In User Interface Software and Technology, pages 161-169. ACM, 1999.
[5] Sachs E., Roberts A., and Stoops D. 3-draw: A tool for designing 3d shapes. IEEE Computer Graphics and Applications, (6):202-211, 1991.
[6] M. Krueger. Artificial Reality II. Addison-Wesley, 1991.
[7] T. Poston and L. Serra. Dexterous virtual work. Commun. ACM, 39(5):37-45, 1996.
[8] L. Cutler, B. Frohlich, and P. Hanrahan. Two-handed direct manipulation on the responsive workbench. In ACM/SIGGRAPH Symposium on Interactive 3D Graphics, pages 107-114. ACM, 1997.
[9] Barry Fowler and Richard Bartels. Constraint-based curve manipulation. IEEE Computer Graphics and Applications, pages 43-49, 1993.
[10] M. J. Banks, E.Cohen, and T. I. Mueller. An envelope approach to a sketching editor for hierarchical free-form curve design and modification. In R. N. Goldman and T. Lyche, editors, Knot Insertion and Deletion Algorithms. 1993, SIAM.
[11] J.M. Zheng, K.W. Chan, and I. Gibson. A new approach for direct manipulation of free-form curves. Graphics Forum, 17(3):C327-C334, 1998.
[12] Les Piegl and Wayne Tiller. The nurbs book. Monographs in Visual Communications, 1995.
[13] T. Grossman, R. Balakrishnan, G. Kurtenbach, G. Fitzmaurice, A. Khan, and W. Buxton. Creating principal 3d curves with digital tape drawing. In Proceedings of the 2002 ACM Conference on Human Factors in Computing Systems, pages 417-423, 2002.
[14] P. Michalik, D. Kim, and B. Bruederlin. Sketch- and constraint-based design of b-spline surfaces. In Seventh ACM Symposium on Solid Modeling and Applications, pages 297-304. ACM, June 2002.
[15] Karan Singh and Eugene Fiume. Wires: a geometric deformation technique. In ACM SIGGRAPH 98, pages 405-414. ACM, 1998.
[16] J. Hoschek and D. Lasser. Fundamentals of computer aided geometric design. A.K. Peters, Ltd., 1989.
[17] G. Golub and C. van Loan. Matrix Computations. The Johns Hopkins University Press, 1989.
[18] Alias/wavefront. MAYA 6.0, 2005.

김 대 현

1994년 국립서울산업대학교 전산과 학사 1995년 고려대학교 컴퓨터학과 대학원 석사. 2004년 독일 Bremen 대학교 공학박사. 1991년~1999년 한국전자통신연구원 연구원. 2004년~현재 (재)그래픽스연구원 선임연구원. 관심분야는 컴퓨터그래픽스

김 명 준

1989년 한국과학기술대학 전산학과 학사 1991년 한국과학기술원 전산학과 석사 1996년 한국과학기술원 전산학과 박사 1996년~1997년 University of Washington 연구원. 1997년~1998년 시스템공학연구소 선임연구원. 1998년~2000년 한국전자통신연구원 선임연구원. 2000년~2001년 ㈜ 엔룩스 연구소장. 2001년~현재 이화여자대학교 디지털미디어학부 조교수. 관심분야는 컴퓨터그래픽스, 물리기반 애니메이션, 게임 프로그래밍