

NURBS Fusion

Xin Liu, Marina Gavrilova, Faramarz Samavati
 Department of Computer Science, University of Calgary
 2500 University Drive NW, Calgary, AB, T2N 1N4, Canada
 liuxin@ucalgary.ca, marina@cpsc.ucalgary.ca, samavati@cpsc.ucalgary.ca

Abstract

In this paper, we propose a new NURBS modeling technique, called NURBS fusion and present a complete algorithm for NURBS fusion. Working on a set of intersecting NURBS surfaces, NURBS fusion automatically eliminates the intersected parts and connects the remaining parts smoothly with auxiliary surfaces, so that a polysurface composed of multiple nicely joined NURBS patches is obtained. The surface fusion algorithm performs in three steps. First, surface/surface intersection is computed by a subdivision based algorithm, which gives intersecting point pairs in parameter domains. Second, base surfaces are trimmed "excessively" along intersection curves to eliminate the intersected parts and make gaps that will be filled by the auxiliary connection surfaces. Finally, bicubic B-spline auxiliary surfaces are constructed to connect the base surfaces in a smooth fashion. Our algorithm is concise, fast, and it produces low degree NURBS results compatible with most data-exchange standards.

1. Introduction

A Non-uniform Rational B-Spline (NURBS) surface in 3D space is a free-form surface defined mathematically on a set of control points. It can be viewed as a deformation of a planar rectangle, which is the parameter domain. Due to the simplicity of the geometry of the planar rectangle, it is very hard to construct a complex 3D model with a single NURBS surface. A usual approach is designing simple parts of the model separately with multiple NURBS surfaces and then plugging them together. But a simple plugging approach produces models with redundant and non-manifold surfaces, which is disadvantageous for most CAD/CAM and computer graphics applications. Although this problem can be partly solved by applying Boolean operations that are supported by some modern NURBS modeling programs, such as RhinoTM, the result often contains the undesirable sharp edges, which need complicated pro-

cedures to be eliminated. Can we obtain a smooth model by assembling the NURBS surfaces with a single procedure? That is the motivation of this presented work.

NURBS fusion is a technique working on a set of intersecting NURBS surfaces. It automatically eliminates the intersected parts and connects the remaining parts in a smooth fashion. The result of NURBS fusion should be a *polysurface* composed of multiple nicely joined NURBS patches.

As a powerful modeling tool, NURBS has received much attention in the literature since it was introduced in 1970's [15]. However, to the best of our knowledge, the concept of NURBS fusion or something similar has never appeared in the literature. Nevertheless, relevant work can be found in the fields of surface/surface intersection (SSI) and connection surface construction.

Computing SSI curves is a fundamental problem of CAD/CAM. Since an exact solution of SSI can be found only for some special surface classes, numerical approximation methods are used for general cases, among which the *subdivision-based* and *marching* algorithms are most popular. In subdivision based algorithms [4, 9, 12, 13], the surfaces are subdivided into a great number of patches and a stack of all possibly intersecting patch pairs is built. The patch pairs are then tested for intersection. Impossible pairs are culled away, and the remaining pairs are either directly used as the intersections [13] or substituted by planar patches to compute piecewise linear intersection curves [12]. The subdivision process is usually accelerated by using a quad-tree structure and subdividing the patches in a coarse-to-fine fashion. Marching algorithms [2, 1] begin by finding a starting point on the intersection curve and then march along the curve to get the successive points. Such algorithms are inefficient to deal with the singular cases, and thus it is very difficult to obtain all intersection segments robustly.

Connection surface is a surface that smoothly connects two given surfaces, called *base surfaces*, along certain curves, called *rail curves*. While most research in the field of connection surface construction has concentrated on con-

necting base surfaces along their boundary curves [11, 18, 3, 10, 6], few works [8, 14, 19] deal with connecting base surfaces along trimming curves. The main reason is that trimming curves, usually defined in parameter domains, often have very high degrees¹, and thus constructing a low degree surface to interpolate those curves is actually impossible. Filip [8] and Kim and Elber [14] both constructed a connection surface with Hermite interpolation. The results are implicitly represented and actually have very high degrees, which do not comply with most data-exchange standards. Zhang and Cheng [19] constructed the connection surface as a set of small blends instead of a single blend. They used one optimization process to fit the base surface to the trimming curves, and used another optimization process to make the connection surface satisfy certain continuity constraints. The algorithm is complex. It produces low degree NURBS results, but the results are generally not pseudo- G^1 continuous [19], because it is not always possible to fit a surface to an arbitrary curve as pointed out by Celniker and Welch [5].

Our algorithm for NURBS surface fusion is composed of three steps, as shown in Fig. 1. First, surface/surface intersection is computed by a subdivision based algorithm, which finds intersecting points in both parameter domains. We designed two special data structures for the SSI algorithm. They make the SSI algorithm efficient and well suited to the following parts of our algorithm. Second, base surfaces are trimmed “excessively” by enlarged intersection curves to eliminate intersected parts and make gaps for the auxiliary surfaces. By changing the enlargement factor, users are able to control the size of each auxiliary surface. Finally, bicubic B-spline auxiliary surfaces are constructed to connect the base surfaces in a smooth fashion. By changing two parameters, users are able to control the tensions of the auxiliary surfaces along u and v (the parameters) directions. As Zhang and Cheng’s work [19], our algorithm produces low degree NURBS results. But we avoid the complex alignment procedure in [19] by constructing structured rail curves and we directly construct high quality connection surfaces by piece-wise cubic Bézier interpolations rather than using optimization processes. So our algorithm is more concise and efficient. Our algorithm ensures that the NURBS patches meet at a finite set of points with (1) the same tangent planes and (2) the same tangent lines of the boundary curves. So the resulting polysurface quickly approaches a perfect solution, which is G^1 continuous everywhere, as the size of the point set increases.

The contributions of this paper include:

- the introduction of NURBS fusion,

¹For a parametric surface with degrees m and n , the 3D trimming curve on surface derived by a Bézier domain curve of degree p will be of degree $(m + n) \times p$ [7].

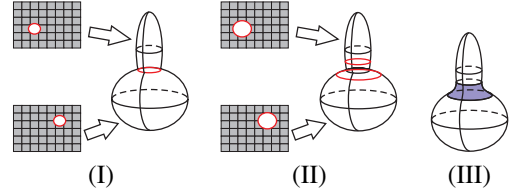


Figure 1. Three steps of our NURBS fusion algorithm.

- a complete algorithm for NURBS fusion,
- two data structures for surface/surface intersection,
- and an algorithm for connection surface construction.

The rest of this paper is organized as follows. We first discuss the surface/surface algorithm in Sec. 2. The problem of base surface trimming with enlarged intersection curves are discussed in Sec. 3. The auxiliary connection surface trimming algorithm is presented in Sec. 4. Experimental results are presented in Sec. 5. Finally we conclude in Sec. 6. We use two ellipsoids as typical NURBS examples through the paper to help explanation.

2 Surface/surface intersection

As the first step, we need to find intersection of two base surfaces. Among various existing SSI algorithms, we use the subdivision-based algorithm because it is robust and it produces results in both parameter domains. Our algorithm subdivides each parameter domain into many small square sub-regions and check if each sub-patch pair, corresponding to each sub-region pair in parameter domains, intersects. Precise intersection test is also a very hard problem. A usual approach is to compute conservative bounding volumes of the sub-patches, and use them instead of the sub-patches for intersection test. The early work of of Hanna et al. [9] uses axis aligned bounding boxes; Houghton and Emmett [11] used oriented bounding boxes; Huber [13] introduced more complicated tight parallelepipeds as bounding volumes. We choose the oriented bounding boxes for its simplicity and efficiency. If the oriented bounding boxes of the two sub-regions A_1 and A_2 intersect with each other, the corresponding sub-patches, $S_1(A_1)$ and $S_2(A_2)$ are taken as *possibly intersecting*; otherwise $S_1(A_1)$ and $S_2(A_2)$ are taken as *non-intersecting*. If two sub-patches are possibly intersecting, we say that they have a *possibly intersecting relationship*.

A brute force algorithm, testing the possible intersection between each pair of small sub-patches, has a very high time complexity. As Houghton et al. [12] did, we use a quad-tree

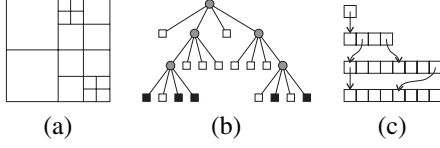


Figure 2. (a) Multi-resolutional sub-regions; (b) logical structure of quad-tree; (c) storage structure of quad-tree.

based multiresolutional strategy to quickly eliminate non-interesting patch pairs. In the first step, we suppose the two input surfaces, corresponding to the root nodes of the quad-trees, are possibly intersecting with each other. Each root node is subdivided two times to make the subregions (in parameter domains) corresponding to the leaf nodes relatively small. In the subdivision process, each child node inherits the possibly intersecting relationships from its parent. Then each possibly intersecting pair is tested for intersection; impossible pairs are culled away. In the next step, each leaf node with possible intersections is subdivided into four nodes. Then each possibly intersecting pair is tested again to cull away impossible pairs. This process is repeated until the size of subregions is small enough.

Inspired by the oc-tree representation [17] in the computer vision field, we designed a special storage structure for quad-tree representation. Fig. 2 shows the multi-resolution structure of subregions in parameter domain, the logical structure and the storage structure of a quad-tree. In this storage structure, each node of the quad-tree is composed of coordinate ranges of the sub-region, a list of possibly intersecting nodes in the same subdivision level of the other quad-tree, and an index to its first child. All nodes in the same level of a quad-tree are stored in the same array. Because all operations are performed in the same level, such storage structure is very efficient in the SSI algorithm.

To represent the possibly intersecting relationships between sub-regions in parameter domains, we also designed a *Possibly Intersecting Image* (PII) structure, as shown in Fig. 3. Let $S_1(u, v)$ and $S_2(u, v)$ stand for two parametric surfaces defined on parameter domains $D_1 = [0, 1] \times [0, 1]$ and $D_2 = [0, 1] \times [0, 1]$, respectively. The SSI algorithm computes two PIIs, I_1 and I_2 of $n_I \times n_I$ pixels: each pixel of the PIIs at (i, j) stands for a rectangular sub-region of the parameter domain $A(i, j) = \left[\frac{i}{n_I}, \frac{i+1}{n_I} \right] \times \left[\frac{j}{n_I}, \frac{j+1}{n_I} \right]$. Each pixel of the images stores a list of possibly intersecting pixels in the other PII, or stores an empty list if there are no possibly intersecting pixels. Formally,

$$\begin{aligned} (i_2, j_2) \notin I_1(i_1, j_1) &\Leftrightarrow (i_1, j_1) \notin I_2(i_2, j_2) \\ &\Rightarrow S_1(A(i_1, j_1)) \cap S_2(A(i_2, j_2)) = \emptyset. \end{aligned} \quad (1)$$

All pixels of a PII is stored in a matrix structure. The advan-

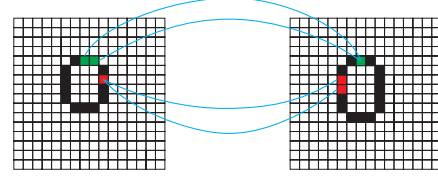


Figure 3. Possibly intersecting images.

tage of the PII structure is that it can be treated by standard image processing techniques, as is discussed in Sec. 3.

With the two data structures described above, the pseudo-code of the SSI algorithm is shown below.

```

SSI(Surface S1, Surface S2, Matrix Img1, Matrix Img2)
// Input: S1 and S2
// Output: Img1 and Img2
{
  Quad-tree qt1, qt2;
  /*Suppose the surfaces intersect with each other*/
  qt1.root.its_list.push(0); qt2.root.its_list.push(0);
  /*Subdivide both patches and
  the possibly intersecting relationship*/
  Subdivide(qt1, qt2);
  for (i = 1; i < DEPTH; i ++){
    Subdivide(qt1, qt2);
    /*Test intersection and cull away
    non-intersecting pairs*/
    CullIntersections(qt1, qt2);
  }
  /*Write out results*/
  SetMatrix (Img1, empty_list);
  SetMatrix (Img2, empty_list);
  for (each node ni of the most detailed layer of qt1)
    if (ni.its_list.size() > 0)
      copy ni.its_list to Img1[ImagePoint(ni)];
  for (each node nj of the most detailed layer of qt2)
    if (nj.its_list.size() > 0)
      copy nj.its_list to Img2[ImagePoint(nj)];
}

```

The SSI algorithm has an asymptotic time complexity of $O(n_{s1} \times n_{s2})$, where n_{s1} and n_{s2} are the numbers of possibly intersecting pixels in the two PIIs respectively. Unlike Boolean operations, NURBS fusion only needs an approximate solution of the intersection curves, because the curves are to be enlarged to some extent anyway. So a moderate resolution like 256×256 is enough for most applications. The SSI algorithm runs very fast in such cases.

3 Base surface trimming

SSI produces two images I_1 and I_2 in which the computed intersections are represented by image regions, as shown in Fig. 4. In real applications, there may be multiple intersection curves on each base surface and each of them may be open or closed. However, in the following text we always suppose that there is only a single closed intersection curve on each base surface and the trimmed region is enclosed by the intersection curve in the parameter domain for the simplicity of description. (These images should be considered as periodic when the surface is closed in u or v directions.) It is not difficult to extend the algorithms to general cases with some careful work. By image processing techniques, the images I_1 and I_2 are segmented into trimmed and preserved regions and contour curves C_1 and C_2 are extracted. The algorithm then walks along the contour, $C_1 \in I_1$, and selects a set of points, $\mathbb{P}_1 = \{p_{1,0}, p_{1,1}, \dots, p_{1,n_C-1}\}$, with equal stride. From \mathbb{P}_1 we find a corresponding point set on C_2 , $\mathbb{P}_2 = \{p_{2,0}, p_{2,1}, \dots, p_{2,n_C-1}\}$, where $p_{2,i}$ is the best matching point of $p_{1,i}$. Here best matching means (1) $p_{2,i} \in C_2$ is one of the nearest points to the possibly intersecting points recorded at $I_1(p_{1,i})$ and (2) $p_{2,i}$ has the shortest distance to $p_{1,i}$ among all nearest points when mapped onto the 3D base surfaces. The trimming curves $T_1(t)$, $T_2(t)$ are constructed as two closed cubic B-spline curves:

$$T_i(t) = \sum_{j=0}^{n_C+2} N_{j,4}(t)P_{i,j} \quad (2)$$

that satisfy

$$\begin{cases} T_i(t_k) = p_{i,k} \\ P_{i,n_C} = P_{i,0} \\ P_{i,n_C+1} = P_{i,1} \\ P_{i,n_C+2} = P_{i,2} \end{cases}, \quad (3)$$

where $i = 0, 1$, $k = 0, 1, \dots, n_C - 1$, and $t_k = \frac{1}{n_C} \cdot k$. In above equations, $N_{j,4}(t)$ are B-spline basis functions and $P_{i,j}$ are control points. $T_1(t)$ and $T_2(t)$ have the same degree, the same number of control points, and the same parameter value produces a pair of corresponding points on the base surfaces. We say they are *co-structured*.

In a next step, each trimming curve is enlarged by moving the points in \mathbb{P}_i away from the geometric center, CG_i , of the interior region by an enlargement factor, ϵ_i :

$$p'_{i,j} = p_{i,j} + |p_{i,j} - CG_i| \cdot \epsilon_i. \quad (4)$$

This is a scaling operation that can be performed on trimming curves with any shapes. In the follow text we still use the notation of $p_{i,j}$ instead of $p'_{i,j}$ to denote the points in \mathbb{P}_i . By enlarging the trimming curves, a gap is created between the trimmed base surfaces for the auxiliary connection surface. The size of the gap is controlled by ϵ_i . Examples of

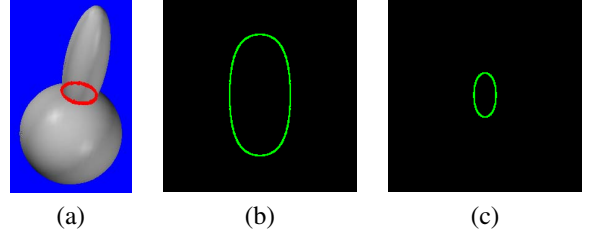


Figure 4. (a) Two intersecting NURBS surfaces. The red ring shows the intersections on surface computed by SSI; (b) intersections in the parameter domain of the lower object; (c) intersections in the parameter domain of the upper object.

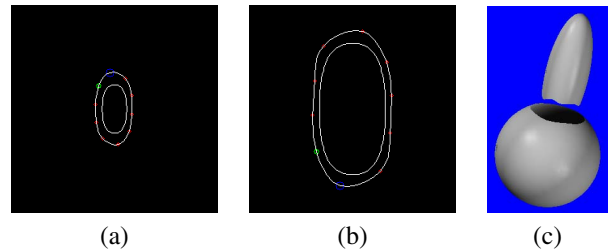


Figure 5. (a) and (b) Trimming curves in parameter domains; (c) trimmed base surfaces.

trimming curves are shown in Fig. 5 (a) and (b), with the corresponding trimmed base surfaces shown in Fig. 5 (c).

4 Auxiliary connection surface construction

Now, two trimmed base surfaces are prepared. We are going to build an auxiliary surface that connects the base surfaces along the trimming curves in a smooth fashion. The 3D trimming curves, $\mathcal{T}_1(t) = S_1(T_1(t))$ and $\mathcal{T}_2(t) = S_2(T_2(t))$, are 3D curves on the surface (CONS) that have very high degrees. Even so, we can compute the derivatives of them as

$$\frac{d\mathcal{T}_i(t)}{dt} = \frac{\partial S_i(\mathcal{T}_i(t))}{\partial u} \cdot \frac{du_{\mathcal{T}_i}(t)}{dt} + \frac{\partial S_i(\mathcal{T}_i(t))}{\partial v} \cdot \frac{dv_{\mathcal{T}_i}(t)}{dt}, \quad (5)$$

where $u_{\mathcal{T}_i}(t)$ and $v_{\mathcal{T}_i}(t)$ stand for the u and v components of $\mathcal{T}_i(t)$; $i = 1, 2$.

Because $\mathcal{T}_1(t)$ and $\mathcal{T}_2(t)$ are high degree curves, it is actually impossible to construct a low degree surface exactly connecting them. Our strategy is constructing a bicubic B-spline surface with the boundaries well approximate $\mathcal{T}_1(t)$ and $\mathcal{T}_2(t)$. For that we first take n_Q points $\mathbb{Q}_i = \{Q_{i,0}, Q_{i,1}, \dots, Q_{i,n_Q-1}\}$ on $\mathcal{T}_i(t)$.

$$Q_{i,j} = \mathcal{T}_i(t_j), \quad (6)$$

where $i = 1, 2$; $t_j = \frac{j}{n_Q}$. $Q_{0,j}$ and $Q_{1,j}$ are corresponding points because $\mathcal{T}_1(t)$ and $\mathcal{T}_2(t)$ are co-structured (see Sec. 3). The connection surface should satisfy (1) its two boundary curves pass through $Q_{0,j}$ and $Q_{1,j}$, (2) its two boundary curves have the same tangent directions as the trimming curves at $Q_{0,j}$ and $Q_{1,j}$ and (3) the connection surface has the same tangent plane as the base surfaces at $Q_{0,j}$ and $Q_{1,j}$. Mathematically, the connection surface can be described as

$$S_{conn}(u, v) = \sum_{i=0}^{n_u-1} \sum_{j=0}^{n_v-1} N_{i,4}(u) N_{j,4}(v) P_{i,j} \quad (7)$$

that satisfies

$$\left\{ \begin{array}{l} S_{conn}(\bar{u}_k, 0) = Q_{1,k} \\ \vec{n}_{conn}(\bar{u}_k, 0) = \varsigma \vec{n}_1(Q_{1,k}) \\ \left. \frac{dS_{conn}(u,0)}{du} \right|_{u=\bar{u}_k} = \xi \left. \frac{d\mathcal{T}_1(t)}{dt} \right|_{t=t_k} \\ S_{conn}(\bar{u}_k, 1) = Q_{2,k} \\ \vec{n}_{conn}(\bar{u}_k, 1) = \varsigma \vec{n}_2(Q_{2,k}) \\ \left. \frac{dS_{conn}(u,1)}{du} \right|_{u=\bar{u}_k} = \xi \left. \frac{d\mathcal{T}_2(t)}{dt} \right|_{t=t_k} \end{array} \right., \quad (8)$$

where $k = 0, 1, \dots, n_Q - 1$; $\vec{n}_{conn}(u, v)$, $\vec{n}_1(u, v)$ and $\vec{n}_2(u, v)$ stand for normal vectors of S_{conn} , S_1 , and S_2 respectively; ς and ξ are constants. In Equ. (7), $N_{i,4}(u)$ and

$N_{j,4}(v)$ stand for cubic B-spline basis functions; $P_{i,j}$ are the control points of the connection surface to be solved.

This is a typical surface fitting problem. Since $S_{conn}(u, v)$ is a tensor product surface, the problem can be solved simply and efficiently by a sequence of curve fittings [16]. For fixed k ,

$$\begin{aligned} Q_{l,k} &= \sum_{j=0}^{n_v-1} N_{j,4}(\bar{v}_l) \left(\sum_{i=0}^{n_u-1} N_{i,4}(\bar{u}_k) P_{i,j} \right) \\ &= \sum_{j=0}^{n_v-1} N_{j,4}(\bar{v}_l) R_{k,j}, \end{aligned} \quad (9)$$

$$R_{k,j} = \sum_{i=0}^{n_u-1} N_{i,4}(\bar{u}_k) P_{i,j}. \quad (10)$$

In above equations, $l = 1, 2$; $\bar{v}_1 = 0$; $\bar{v}_2 = 1$; $R_{k,0}, R_{k,1}, \dots, R_{k,n_v}$ are control points of the v -curves. Now fixing j and allowing k vary, Equ. (10) is a curve interpolation through points $R_{0,j}, R_{1,j}, \dots, R_{n_Q-1,j}$, where $P_{0,j}, P_{1,j}, \dots, P_{n_u,j}$ are the computed control points. The procedures of the algorithm to obtain all $P_{i,j}$ are shown below:

1. Do n_Q curve interpolation through $Q_{0,k}$ and $Q_{1,k}$ (in v direction) with proper derivatives at $Q_{0,k}$ and $Q_{1,k}$; this yields $R_{k,j}$, $k = 0, 1, \dots, n_Q - 1$, $j = 0, 1, \dots, n_v$.
2. Do n_v curve interpolations (in u direction) through points $R_{0,j}, R_{1,j}, \dots, R_{n_Q-1,j}$ with proper derivatives at $R_{0,j}, R_{1,j}, \dots, R_{n_Q-1,j}$, this yield $P_{i,j}$, $i = 0, 1, \dots, n_u$, $j = 0, 1, \dots, n_v$.

4.1 Interpolation in v direction

v -curve $\mathcal{V}_k(v) = S_{conn}(\bar{u}_k, v)$ interpolates $Q_{1,k}$ and $Q_{2,k}$. It is perpendicular to the normal of $S_1(u, v)$ at $Q_{1,k}$ and perpendicular to the normal of $S_2(u, v)$ at $Q_{2,k}$, where $k = 0, 1, \dots, n_Q - 1$. We use a cubic Bézier curve for $\mathcal{V}_i(v)$, $i = 0, 1, \dots, n_Q - 1$. So $n_v = 4$; the knot sequence is

$$V = 0, 0, 0, 0, 1, 1, 1, 1. \quad (11)$$

Let

$$\left\{ \begin{array}{l} \mathcal{V}_k(0) = Q_{1,k} \\ \mathcal{V}'_k(0) = \varsigma' \vec{t}_{1,k} \\ \mathcal{V}_k(1) = Q_{2,k} \\ \mathcal{V}'_k(1) = \varsigma' \vec{t}_{2,k} \end{array} \right., \quad (12)$$

where

$$\begin{aligned} \vec{t}_{1,k} &= \vec{n}'_1 \times \vec{n}_1 \\ &= \left(\text{unit}(\vec{n}_1(Q_{1,k})) \times \text{unit}(Q_{2,k} - Q_{1,k}) \right) \\ &\quad \times \text{unit}(\vec{n}_1(Q_{1,k})), \end{aligned} \quad (13)$$

as shown in Fig. 6 (a); ς' is a constant. $\vec{t}_{2,k}$ is computed similarly.

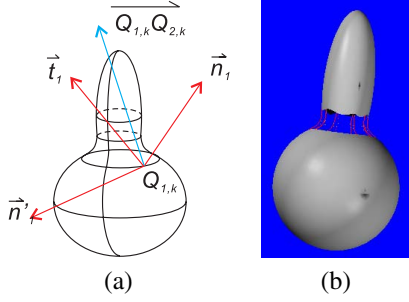


Figure 6. (a) Computation of the derivative of v -curve; (b) results of interpolation in v direction.

Control points of $\mathcal{V}_k(v)$, $R_{k,j}$ can be constructed directly:

$$\begin{cases} R_{k,0} = Q_{1,k} \\ R_{k,1} = Q_{1,k} + \zeta'' \vec{t}_{1,k} \\ R_{k,2} = Q_{2,k} - \zeta'' \vec{t}_{2,k} \\ R_{k,3} = Q_{2,k} \end{cases} \quad (14)$$

where ζ'' is a parameter with which user can control the tension of the connection surface. An example of v -curves constructed by the above algorithm is shown in Fig. 6 (b).

4.2 Interpolation in u direction

The cubic B-spline curves $\mathcal{U}_j(u)$, $j = 0, 1, 2, 3$, interpolate $R_{k,j}$, $k = 0, 1, \dots, n_Q - 1$. $\mathcal{U}_0(u)$ and $\mathcal{U}_3(u)$ have the same tangent directions as $\mathcal{T}_1(t)$ and $\mathcal{T}_2(t)$ at $R_{k,0}$ and $R_{k,3}$ respectively. We use piecewise cubic Bézier curves for $\mathcal{U}_j(u)$. So $n_u = 3n_Q$; the knot sequence is

$$U = \{ 0, 0, 0, 0, u_1, u_1, u_1, u_2, u_2, u_2, \dots, u_{n_Q-1}, u_{n_Q-1}, u_{n_Q-1}, 1, 1, 1, 1 \}, \quad (15)$$

where

$$u_k = \frac{\sum_{r=0}^{k-1} |Q_{1,(r+1) \bmod n_Q} - Q_{1,r}|}{\sum_{r=0}^{n_Q-1} |Q_{1,(r+1) \bmod n_Q} - Q_{1,r}|}, \quad (16)$$

for $k = 1, 2, \dots, n_Q - 1$. Let

$$\begin{cases} \mathcal{U}_j(u_k) = R_{k,j} \\ \mathcal{U}_j^+(u_k) = \xi' \vec{\tau}_{k,j} \\ \mathcal{U}_j(u_{k+1}) = R_{(k+1) \bmod n_Q, j} \\ \mathcal{U}_j^-(u_{k+1}) = \xi' \vec{\tau}_{(k+1) \bmod n_Q, j} \end{cases}, \quad (17)$$

for $k = 0, 1, \dots, n_Q - 1$, $j = 0, 1, 2, 3$. In above equation, ξ' is a constant. The tangent directions of $\mathcal{U}_0(u)$ and $\mathcal{U}_3(u)$ equal to the tangent directions of the trimming curves at $R_{k,0}$ and $R_{k,1}$, which can be computed by Equ. (5). We can add two additional constraints to let the tangent directions of

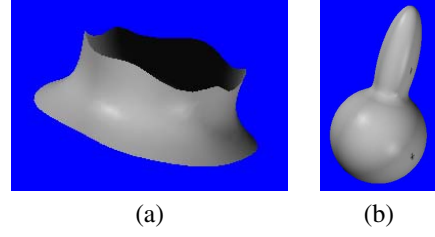


Figure 7. (a) Auxiliary connection surface; (b) the polysurface computed by NURBS fusion.

$\mathcal{U}_1(u)$ and $\mathcal{U}_2(u)$ to be the linear interpolation of the tangent directions of $\mathcal{U}_0(u)$ and $\mathcal{U}_3(u)$. So we have

$$\begin{cases} \vec{\tau}_{k,0} = \text{unit}(\mathcal{T}'_1(t_k)) \\ \vec{\tau}_{k,1} = \alpha_k \text{unit}(\mathcal{T}'_1(t_k)) + (1 - \alpha_k) \text{unit}(\mathcal{T}'_2(t_k)) \\ \vec{\tau}_{k,2} = \beta_k \text{unit}(\mathcal{T}'_1(t_k)) + (1 - \beta_k) \text{unit}(\mathcal{T}'_2(t_k)) \\ \vec{\tau}_{k,3} = \text{unit}(\mathcal{T}'_2(t_k)) \end{cases}, \quad (18)$$

where

$$\begin{cases} \alpha_k = \frac{|R_{k,3} - R_{k,1}|}{|R_{k,1} - R_{k,0}| + |R_{k,3} - R_{k,1}|} \\ \beta_k = \frac{|R_{k,3} - R_{k,2}|}{|R_{k,2} - R_{k,0}| + |R_{k,3} - R_{k,2}|} \end{cases}. \quad (19)$$

Control points of $\mathcal{U}_j(u)$ can be constructed directly:

$$\begin{cases} P_{3k,j} = R_{k,j} \\ P_{3k+1,j} = R_{k,j} + \xi'' \vec{\tau}_{k,j} \\ P_{3k+2,j} = R_{(k+1) \bmod n_Q, j} - \xi'' \vec{\tau}_{(k+1) \bmod n_Q, j} \end{cases}, \quad (20)$$

where $k = 0, 1, \dots, n_Q - 1$; $j = 0, 1, 2, 3$; and ξ'' is a parameter with which user can control the tension of the connection surface. An example of auxiliary connection surface constructed by interpolating along u direction is shown in Fig. 7.

5 Experiments

In this section, we demonstrate the performance of our algorithm by a series of experiments. The running time reported was measured on a laptop computer with an Intel Pentium T2080 CPU and 2Gb RAM. The resolution of PIIs was set to 256×256 pixels.

As can be known from the auxiliary connection construction process detailed in Sec. 4, our algorithm does not ensure G^1 continuity along intersection curves. That means cracks, or discontinuities, may occur along the boundaries of NURBS patches constituting the polysurface. However, tangent plane continuity is guaranteed at a finite set of points along the boundaries, as can be known from Equ. (8), so that we can expect to improve the quality of the polysurface by increasing the size of the \mathcal{Q}_i . While quantitative

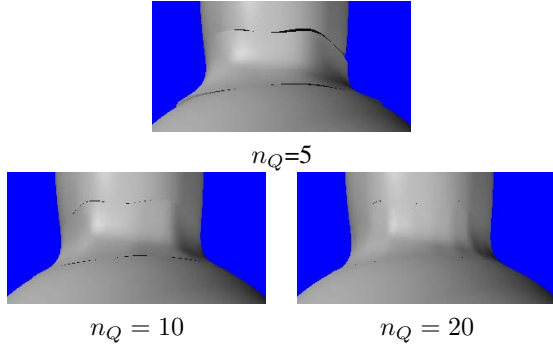


Figure 8. NURBS fusion with different precisions.

analyses can only be done in a case by case fashion, here we show the effects intuitively by experiments on fusing two ellipsoids. Fig. 8 shows the resulting polysurfaces produced by $n_Q = 5$, $n_Q = 10$ and $n_Q = 20$ respectively. From the images we can see the cracks between connection and base surfaces diminish quickly as n_Q increases from 5 to 20. The reason behind is our connection surface construction algorithm guarantees the boundary curves have the same tangent lines in addition to the surfaces have the same tangent planes at any point in Q_i . The running times are all 0.203s. Changes of n_Q don't affect the running time obviously.

To show how the surface patches are connected, we present another experimental result on fusing a free-form surface and an ellipsoid in Fig. 9. Because the free-form surface is not closed, we are able to observe the fusion parts from both sides. In this experiment, we set $n_Q = 20$. From Fig. 9 we can clearly see that after processing by our algorithm, the polysurface looks as good as a single patch of smooth surface. The running time is 0.265s.

As discussed in Sec. 3, our algorithm allows users to control the shape of the resulting model by changing the enlargement factors. Generally speaking, a big factor makes a large gap between trimmed base surfaces and thus makes a smooth connection surface; a small factor makes a small gap and thus produces a fused model more like produced by a Boolean union operation. Fig. 10 shows NURBS fusion with three pairs of enlargement factors.

Our algorithm also allows users to control the tension of the connecting surface in v and u directions by adjusting ζ'' and ξ'' . Generally speaking, ξ'' can be set to a proper value so that the boundary curves of the connecting surfaces are well fitted to the trimming curves. $\xi'' = 0.5$ is empirically a good choice. Different ζ'' may produce different connection surfaces satisfying different applications. Fig. 11 shows results of surface fusion with three ζ'' values. For clarity, mesh models are shown as well. From the images we can see that a small ζ'' makes big curvatures near to the bound-

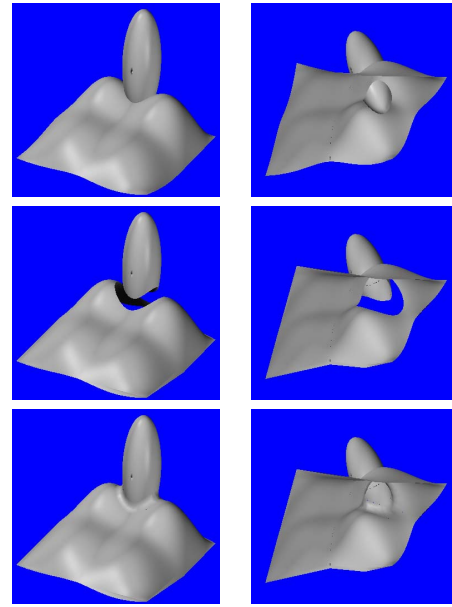


Figure 9. Fusing a free-form surface and an ellipsoid. The first row shows the model before fusion; the second row shows the trimmed base surfaces; the third row shows the model after fusion.

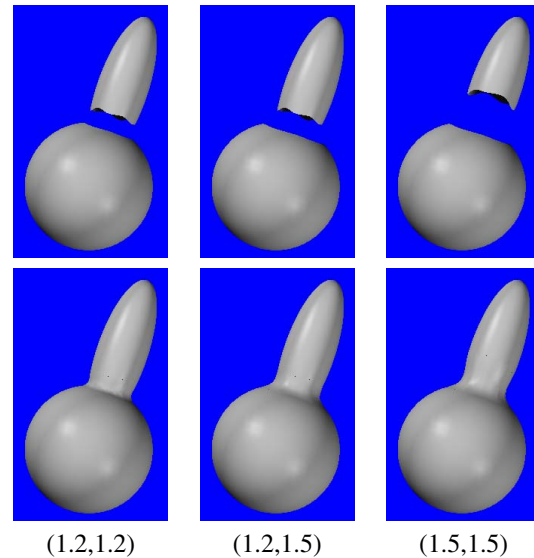


Figure 10. NURBS fusion with different size of gaps. Enlargement factors of the lower and upper surfaces are given below each pair of images.

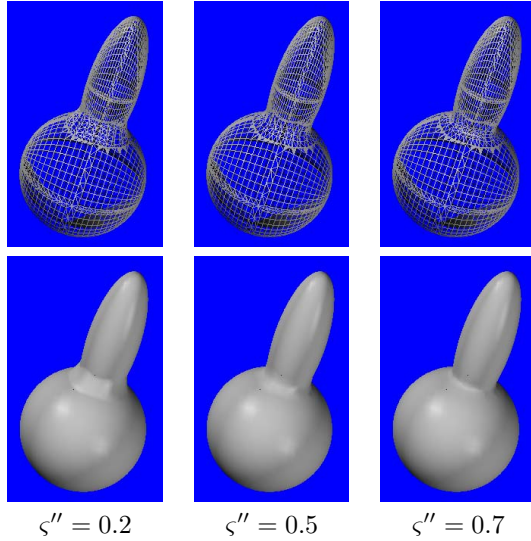


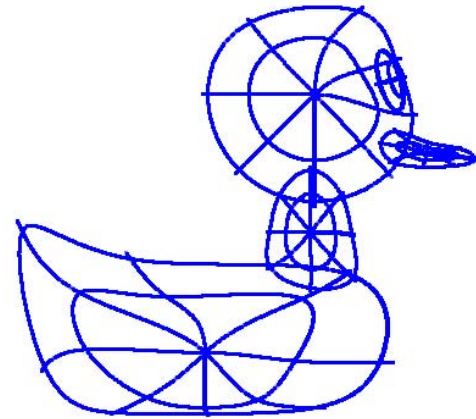
Figure 11. NURBS fusion with different tensions.

ary curves and produces a planar connection surface; a big ζ'' makes big curvatures near to the middle of the auxiliary connection surfaces; a moderate ζ'' produces a very smooth connection surface with curvatures evenly distributed.

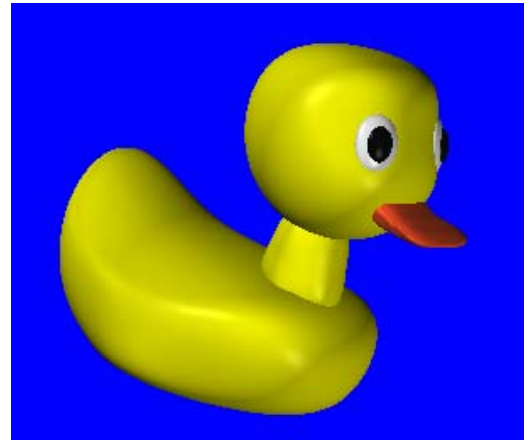
Finally, we present the result of a more complex experiment which constructs a duck model with our NURBS fusion algorithm. The model is composed of 8 NURBS surfaces, as shown in Fig. 12 (a) & (b). Three surface fusions are used to fuse the body and neck, neck and head, head and beak respectively. The polysurface produced by our algorithm is shown in Fig. 12 (c). From the image we can see that the NURBS surface patches are very nicely fused together. The smooth transitions between NURBS patches make the neck and the beak look very natural. The running time is 1.70s.

6 Conclusion

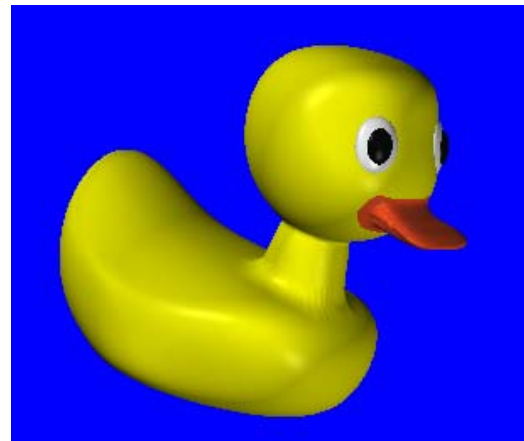
In this paper, we have proposed a new NURBS modeling technique, named NURBS fusion and presented a complete algorithm for NURBS fusion. Given a set of intersecting surfaces, our algorithm first computes the intersection curves. Then the base surfaces are trimmed excessively along the intersection curves. Finally, auxiliary connection surfaces are constructed by piece-wise cubic Bézier interpolations to connect the base surfaces in a smooth fashion. The result of our algorithm is a polysurface composed of multiple low degree NURBS patches, which complies with most data exchange standards. Our algorithm ensures that the NURBS patches meet at a finite set of points with (1) the same tangent planes and (2) the same tangent lines of



(a)



(b)



(c)

Figure 12. Constructing a duck model by NURBS fusion. (a) & (b) Before fusion; (c) after fusion.

the boundary curves.

References

- [1] C. L. Bajaj and G. Xu. Nurbs approximation of surface/surface intersection curves. *Advances in Computational Mathematics*, 2(1):1–21, 1994.
- [2] R. Barnhill and S. Kersey. A marching method for parametric surface/surface intersection. *CAGD*, 7:257–280, 1990.
- [3] A. Bien and F. Cheng. A blending model for parametrically defined geometric objects. In *1st ACM Symposium on Solid Modeling and Applications*, pages 339–347, 1991.
- [4] W. E. Carlson. An algorithm and data structure for 3d object synthesis using surface patch intersections. *ACM SIGGRAPH Computer Graphics*, 16(3):255–263, 1982.
- [5] G. Celniker and W. Welch. Linear constraints for deformable b-spline surfaces. In *Symp. on Interactive 3D Graphics*, pages 165–170. ACM, 1992.
- [6] X. Che, X. Liang, and Q. Li. G1 continuity conditions of adjacent nurbs surfaces. *Computer Aided Geometric Design*, 22(4):285–298, 2005.
- [7] G. Farin. *Curves and Surfaces for CAGD, A Practical Guide*. Morgan Kaufmann Publishers, fifth edition, 2001.
- [8] D. J. Filip. Blending parametric surface. *ACM Trans. Graphics*, 8(3):164–173, 1989.
- [9] S. Hanna, J. Abel, and D. GreenBerg. Intersection of parametric surfaces by means of lookup tables. *IEEE Computer Graphics*, 3(7), 1983.
- [10] E. Hartmann. Gn-blending with rolling ball contact curves. In *Proceedings of the Geometric Modeling and Processing 2000*, pages 385–389. IEEE Computer Society, 2000.
- [11] C. Hoffmann and J. Hopcroft. Automatic surface generation in computer aided design. *The Visual Computer*, 1(4):92–100, 1985.
- [12] E. Houghton, R. Emmett, J. Factor, and C. Sabharwal. Implementation of a divide-and-conquer method for intersection of parametric surfaces. *CAGD*, 2:171–183, 1985.
- [13] E. H. Huber. Intersecting general parametric surfaces using bounding volumes. In M. Soss, editor, *Proceedings of the 10th Canadian Conference on Computational Geometry*, pages 52–53, Montréal, Québec, Canada, 1998.
- [14] K. Kim and G. Elber. A symbolic approach to freeform parametric surface blends. *J. of Visualization and Computer Animation*, 8(2):69–80, 1997.
- [15] L. Piegl. On nurbs: a survey. *IEEE Computer Graphics and Applications*, 11:55–71, Jan. 1991.
- [16] L. Piegl and W. Tiller. *The NURBS Book*. Springer-Verlag, New York, NY, 2nd edition, 1997.
- [17] R. Szeliski. *Real-Time Octree Generation from Rotating Objects, Technical Report 90/12*. Digital Equipment Corporation, Cambridge Research Lab, 1990.
- [18] J. Warren. Blending algebraic surfaces. *ACM Trans. Graphics*, 8(4):263–278, 1989.
- [19] P. Zhang and F. Cheng. Smooth trimmed nurbs surface connection with tension control. In *Proc. of Geometric Modeling and Processing*, pages 56–65, 2004.