CENTER FOR COMPUTATIONAL MATHEMATICS REPORTS

University of Colorado at Denver
P.O. Box 173364, Campus Box 170
Denver, CO 80217-3364

Fax: (303) 556-8550
Phone: (303) 556-8442
http://www-math.cudenver.edu/

January 1997 UCD/CCM Report No. 97

New Algorithms for Modeling of Curves and Surfaces

Michael Zuev

NEW ALGORITHMS FOR MODELING OF CURVES AND SURFACES

Dr. Michael Zuev, University of Colorado at Denver

Abstract

This article concerns some aspects of curves and surfaces modeling field with regards to spline - interpolation problems. Derivatives computation methods are discussed, such as Modified - Lagrange technique, and Spline-Matrix inversion methodologies for Degrees = 3 and 5. Spline - curve fitting algorithm is proposed, based on the new analytical method for segment fitting with polynomials. New analytical technique for arc-length fast computation is described. Spline - Curves' converters between Hermit, Bezier, B-spline and Power-basis representations are discussed. Spline - Surfaces' construction and evaluation algorithms are presented, such as: evaluation technique for the single u-v patch, conversion between various Spline - representations and partial derivatives' computation algorithms. New Loft - Surfaces' construction and re-parametrization algorithm, free from false loops appearance is proposed, instead of conventional iso-curves lofting approach. New methodology is discussed to create a universal converter from any parametrical surface to the optimized Spline, which keeps tolerance specified and minimizes the amount of u-v patches. New Local Surface Deformation approach is presented.

We will discuss some peculiarities of spline technique, starting from curves as a basis for surfaces. Most of things which will be represented here are implemented in C++ classes and can be easily adapted for relative Projects.

Spline - curves interpolation technique

Let's consider a very conventional Degree = 3 (D3) Spline-segment # n in Hermit - representation:

$$\mathbf{P}(u) = \mathbf{P}_n * H_{03}(t) + \mathbf{P}_{n+1} * H_{33}(t) + \Delta_n * [\, \mathbf{P}_n' * H_{13}(t) + \mathbf{P}_{n+1}' * H_{23}(t) \,]$$

$$t = (u - u_n) / \Delta_n ; \qquad \Delta_n = u_{n+1} - u_n ;$$

Here $H_{mn}(t)$ are Hermit polynomials. To compute the curve at any parameter 'u' we need points $\mathbf{P}$ and derivatives $\mathbf{P}'$ at the segment's ends.

Here and later we consider two kinds of interpolation problems, depending on available input information:

1. We may already have *curve* (meaning - we have an evaluator for points and derivatives) but for any reason we need to convert this curve into the spline - representation. This problem will be discussed later.

2. For many other kinds of tasks in surface modeling field we do NOT have a *curve.* We ONLY have through-points, we do NOT have derivatives at the inner points. For such kind of tasks we have to find:
   a) segments - parametrization (if it is NOT given as an input); and
   b) derivatives at all the segments' ends;
   We will discuss the last problem now.

==========

There are many ways to determine the derivatives. We consider two of them [1].

Modified-Lagrange technique:

$$\mathbf{P}(u) = \sum_{m=0}^{m=M} \mathbf{P}_m * \prod_{n \neq m}(u - u_n) / \prod_{n \neq m}(u_m - u_n) ;$$

Here the derivatives computation in each point is based on direct interpolation between some of the surrounding points. This technique is slightly different for symmetrical, non - symmetrical and periodical cases (we make periodical curve, when Last-point coincides with Start-point).

This method is very fast because, during the computation procedure for each point we can memorize many blocks to use them for the next point's computation. Big Degrees, such as 6, 8 and so on provide high precision for smooth curve interpolation, but the correspondent polynomials are very sensitive to the points' fluctuations. Small Degrees, such as 2 and 4 give less precision, but their polynomials are more stable. They do NOT cause oscillations, when a fluctuation of any point effects the whole curve.

==========

The best solution of this problem can be provided by another technique.

Spline-Matrix inversion methodologies

This approach is much more powerful and reliable than Modified-Lagrange algorithm. First we consider the conventional case: *Degree = 3* (D3).

To find the unknown 1-st derivatives in the inner points we put a specific condition for the 2-nd derivatives. The most popular are two kinds of such conditions: the 2-nd derivatives continuity: $\mathbf{P}''(u_n - 0) = \mathbf{P}''(u_n + 0)$; OR minimization of curvature's integral:

$$\sum \int_{u_n}^{u_{n+1}} du * | \mathbf{P}''(u) |^2 = \min;$$

Curvature's minimization can sometimes be preferable to reduce oscillating troubles. The bad news about this is - you are losing C2-continuity. The only case when these conditions are equivalent is the case of uniform parametrization.

Our goal is to provide any of the 2-nd derivatives' property, adjusting the 1-st derivatives. For both conditions the corresponding equation contains two derivatives from left-segment and two derivatives from right-segment. The central derivative is the same. It brings us to the equation for each inner derivative, being surrounded by two neighboring unknown derivatives:

$$\alpha_n * \mathbf{P}'_{n-1} + \beta_n * \mathbf{P}'_n + \gamma_n * \mathbf{P}'_{n+1} = \mu_n * (\mathbf{P}_n - \mathbf{P}_{n-1}) + \nu_n * (\mathbf{P}_{n+1} - \mathbf{P}_n);$$

Here all the coefficients (α, β, γ, μ, ν) depend on left and right parameter - lengths. For the curve-ends derivatives' computation many kinds of end - conditions may be used. All of them tie the derivatives at 3 neighboring end - points. Such consideration brings us to the 3-diagonal matrix equation for the derivatives' vector:

$$\begin{pmatrix} \alpha_0 & \beta_0 & \gamma_0 & 0 & 0 & 0 \ldots \\ \alpha_1 & \beta_1 & \gamma_1 & 0 & 0 & 0 \ldots \\ 0 & \alpha_2 & \beta_2 & \gamma_2 & 0 & 0 \ldots \\ 0 & 0 & \alpha_3 & \beta_3 & \gamma_3 & 0 \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \end{pmatrix} * \begin{pmatrix} \mathbf{P}'_0 \\ \mathbf{P}'_1 \\ \mathbf{P}'_2 \\ \mathbf{P}'_3 \\ \ldots \end{pmatrix} = \begin{pmatrix} \mu_0*(\mathbf{P}_1 - \mathbf{P}_0) + \nu_0*(\mathbf{P}_2 - \mathbf{P}_1) \\ \mu_1*(\mathbf{P}_1 - \mathbf{P}_0) + \nu_1*(\mathbf{P}_2 - \mathbf{P}_1) \\ \mu_2*(\mathbf{P}_2 - \mathbf{P}_1) + \nu_2*(\mathbf{P}_3 - \mathbf{P}_2) \\ \mu_3*(\mathbf{P}_3 - \mathbf{P}_2) + \nu_3*(\mathbf{P}_4 - \mathbf{P}_3) \\ \ldots \end{pmatrix}$$

The solution of such system require to use the well-known Spline-Matrix inversion procedure. Here it is important to emphasize two things. 1-st: this procedure is stable. 2-nd: the inversion algorithm' performance is proportional to the number of points, while the general matrix inversion algorithm is proportional to the cube of this number.

Also we would like to mention that such procedure can be divided in two steps. The 1-st step prepares everything in arrays concerning inversion of coefficients (α, β, γ, μ, ν). The 2-nd step uses these arrays for direct computation of the derivatives from given points. This is a very time - saving approach because for many tasks in surface modeling field we have to use the same set of coefficients (α, β, γ, μ, ν) but different set of 'points' **P** (for example, when we run though set of iso-curves with unified parametrization, or use cross - partials as 'points' to find twists).

For the periodical case this matrix structure is slightly different. The non-zero elements appear-right-up and left-down, so the Spline-Matrix inversion algorithm has to be slightly modified.

We also have to mention that for many problems of the non - linear physics field we cannot use such Special-Matrix inversion directly. The point is - we must provide numerical stability of this algorithm. For Spline-Matrix inversion the set of coefficients (α, β, γ) is not arbitrary - they satisfy the special stable-condition, which provides numerical stability. However, for non - linear tasks these coefficients depend on unknown function values. It makes the inversion algorithm unstable. This problem was solved when a specific procedure was developed to make such 3 - diagonal matrix inversion absolutely stable even when the stable-condition for the coefficients (α, β, γ) is not fulfilled.

Now we discuss more complicated environment [3]: *Degree = 5* (D5), based on Hermit - representation for each segment of the curve:

$$\mathbf{P}(u) = \mathbf{P}_n * H_{05}(t) + \mathbf{P}_{n+1} * H_{55}(t) + \Delta_n * [\, \mathbf{P}_n' * H_{15}(t) + \mathbf{P}_{n+1}' * H_{45}(t) \,] + \Delta_n^2 * [\, \mathbf{P}_n'' * H_{25}(t) + \mathbf{P}_{n+1}'' * H_{35}(t) \,] ;$$

$$t = (u - u_n) / \Delta_n ; \qquad \Delta_n = u_{n+1} - u_n ;$$

Here to compute the curve at any parameter 'u' we need points **P** and *two* derivatives ($\mathbf{P}'$ and $\mathbf{P}''$) at the segment' ends. It is more tedious than D3 interpolation, but there are several important reasons to use D5 - Splines for curves and surfaces.

The 1-st, major reason is **Continuity** problem: D5-Splines allow us to keep C2-continuity for *Surfaces* (this means the curvatures and all the 2-nd derivatives will be continuous). Within D3-environment this important property can be provided for *Curves,* as we discussed before, but NOT for surfaces. All the CAD developers know that most of intersectors' algorithms and other applications use the 2-nd derivatives, which have to be continuous [2]. Any C2-discontinuity causes unpredictable trouble with intersectors' algorithms in seeds' search and tracer's procedures. Also, rendering and shading are very sensitive to any discontinuities. Sometimes it causes the false blink appearance, but of course, the intersectors' trouble are much more important.

The 2-nd reason is **Accuracy**: choosing D5 instead of D3 we can provide thousands times more precision for the fitting process. It allows us to reduce many times the amount of segments which we need for curve fitting to keep the same tolerance. For *surface*-interpolation it allows the reduction of many dozen times the amount of patches. It is very helpful to save memory - sometimes this is important.

The 3-rd reason is **Effectiveness**: in most CAD - systems we have 1-st and 2-nd derivatives, and partials directly from all evaluators. Within D3 environment the 2-nd derivatives are not in use for interpolation, and it is very sad - we loose the useful information available. Moreover, if we use the 2-nd derivatives to create D5C2-Spline - we have *direct* Hermit representation, so the Spline-Matrix inversion is NOT necessary any more. It allows to increase radically the interpolation performance.

However, the situation becomes much more complicated if for any reason within D5-environment we have ONLY points (NO derivatives). In such a case, to provide interpolation we have to convert two 3-diagonal matrix simultaneously for the 1-st and 2-nd derivatives.

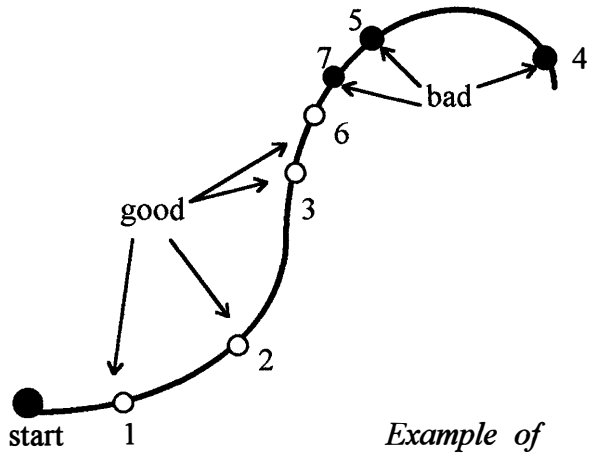
==========

Spline-curves fitting algorithm

Now we will discuss an important fitting problem for curves: how to minimize the amount of non-uniform segments on the original curve to create D5C2-Spline, when the tolerance is specified between the original curve and spline. Intuitively, the more curvature of the region, the less segment - length still keeps the tolerance. However, computer has to provide an algorithm, not intuition.

First we must create a testing procedure for the single segment, which checks - if the segment is good enough for fitting. Here when we say: 'good-segment' - it means, that maximum of deviation from spline-segment to the original-segment is less then tolerance. Let's suppose we have such testing function.

We can start fitting algorithm from a very small segment on the original curve. We will expand or reduce the size of segment after investigation - if it is a 'good-segment' or not ? First we make a polynomial segment's representation based on points and derivatives at the segment-ends, because we have them from the curve's evaluator. Now we use the testing function, and check: does this spline representation fit the original-curve segment ? If yes - we say, this is a 'good-segment' and try to expand it. If no - we cut it twice. If we continue such process - we come to 'good-segment' and 'bad-segment' having the same start-point, but different end-points. Our goal is to find a 'new-good-segment', which lies between 'good' and 'bad' and has maximum possible length. We find it by simple iterative process. Now we have a furthest point from the start, which forms the 'good-segment'. We mark this point as a new start and continue such process until the curve ends. A very last segment in such algorithm may become occasionally small - for this case we simply adjust last two segments to improve the uniformity.



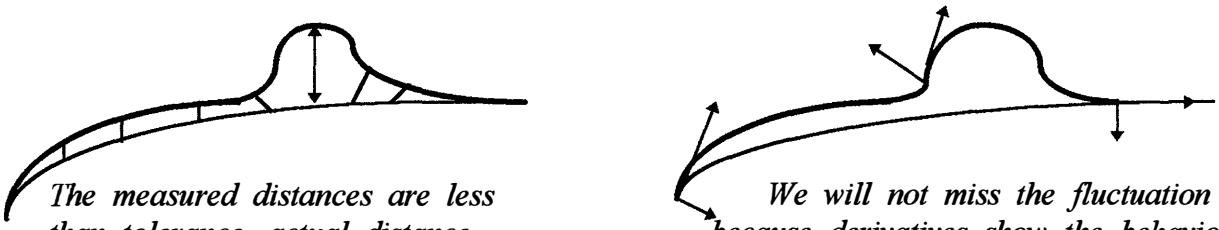
Example of curve - fitting algorithm

This fitting algorithm works very good. It allows to make D5C2-spline from any curve with any big curvature and minimize the amount of segments, which we need to keep the tolerance specified.

==========

The central piece of such algorithm is a testing procedure for the single segment fitting.

Analytical method for segment fit testing



The measured distances are less than tolerance, actual distance is more than tolerance.

We will not miss the fluctuation because derivatives show the behavior

Illustrations for segment fit testing.

There is no fast and reliable numerical algorithm for arbitrary segment testing. The conventional way is to compute many distances from one curve to another. For this purpose we have to find perpendiculars to the original curve, but such way has the well-known disadvantages:

1. It is not reliable, until we do not know a good initial approximation.
2. It works extremely slow because of many probes are necessary.
3. We have a big probability to miss a fluctuation on the original curve.

Another way which we discuss now is free from these disadvantages and consists on 3 steps:

1. We construct D5-polynomial segment (total-spline) using point and both derivatives at each end of segment. This information is available from original curve evaluator;

2. We take just one (!) middle point (and two derivatives) from original curve and construct two (left & right) spline sub-segments;

3. We compare analytically the total-spline with both sub-splines and determine the maximum deviation.

This way is incomparably faster then conventional way and much more reliable because derivatives show us the tendency for any fluctuation. The analytical part of this algorithm finds the maximum of the difference between two D5-polynomials. Generally to determine an extreme of D5 polynomial we have to solve a quartic equation. Fortunately, our case is very specific and simplifies to the quadratic equations, so the analytical solution means the direct formula. It is easy to show how it works: when we use the same derivatives at the left-end for the total-spline and left sub-spline - it means that the corresponding terms of their polynomial decomposition will coincide. This means that the difference between total-spline and left sub-spline will NOT contain the first 3 terms, only the exponents 3, 4 and 5 are presented in the formula: $\Delta\mathbf{P} = \mathbf{a} * t^3 + \mathbf{b} * t^2 + \mathbf{c} * t$;
The same situation repeats on the right-end. To find a corresponding maximum we differentiate such polynomials and come to the *quadratic* equations. This simple analytical procedure gives us the maximum of deviations between total-spline and sub-splines. Comparison of these maximums with tolerance specified brings us to the decision - can this segment be described as D5-polynomial or not ?

There is one more thing we will discuss now with regards to the fitting problem. The point is - for many cases we have to take a special care about the curves' parametrization. Otherwise we may lose most of the improvements.

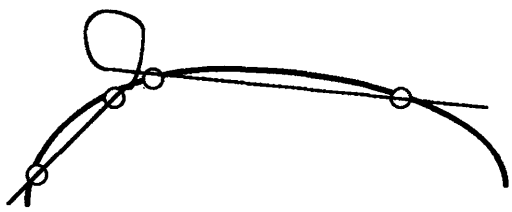
Arc-length fast computation problem

There are many ways [3] for spline-curve parametrization, such as uniform and non-uniform, chord-length and arc-length. It is very important to notice that *any* kind of non-linear re-parametrization will *change* the actual curve ! However, many tasks in the curve and surface modeling field require to change parametrization.

Concerning the fitting problem, our goal was to create the universal program for conversion from arbitrary curve to spline. Here we must consider the cases, where the parametrization of original input curve is not good enough (for example, uniform case), or if we need a specific parametrization for the output spline-curve. Within the iterative fitting algorithm, which we discussed before, one can think that a simplest way is the chord-length re-parametrization. However this way makes our algorithm unstable ! It is easy to show this phenomenon. Indeed, to keep the tolerance we need to maneuver by the through-points on the curve (to optimize the segments-ends). Each change of a through-point on the curve will change the surrounding chord-lengths, so the parametrization will be slightly modified. It causes small changes of actual curve. However these small changes are comparable with tolerance ! It causes the unpredictable diversion of the iterative process, even in the regions, where the curvature is not big, that was tested by many numerical experiments.

The natural way to avoid such troubles is the arc-length parametrization usage, because arc-length fluctuations almost do not depend on the points manipulating. This way provides a very fast conversion for the interpolating iterative processes.

There are many other reasons to try to avoid unnatural parametrization. For example, if we make a curve from a set of points, we usually provide natural parametrization like chord-length or arc-length. For such cases we are practically guaranteed from false loops appearance. However sometimes for whatever reason we can make unnaturally big parameter - length between neighboring points, instead of naturally small distance between them. In this case the interpolating curve will follow unnaturally big parameter-length. The result is - a false loop appearance. Of course, it can be easily shown analytically, but geometrical interpretation is preferable here. Such things occur when a surface construction algorithm is not self-consistent for perpendicular - parameter directions. The best example is a conventional algorithm for loft-surface construction, which we will discuss later.



Example of a false loop appearance in case of uniform parametrization.

One more reason to have unified parametrization for all the curves is a necessity to make a special procedure of end-to-end smoothing for two contacting curves, when their parametrizations are different.

The best way to remove all such problems is to provide the standard parametrization for all curves. The only such standard is arc-length parametrization - it is the natural standard ! Besides, many geometrical formulas and properties become much easier for use in this case (everything concerning curvatures, offset' derivatives, etc.).

One can ask: what's the problem, every mass-property package has the arc-length routine, using Legander polynomials. Yes, we can use it, but not through the interpolation process, while we re-calculate the arc-lengths for each iteration. The reason is simple: mass-property routines require a huge amount of sub-segments to keep the tolerance we want. If we use such routine on every step of iterative process - we will 'die'. This means that we need a direct formula for arc-length computation. Such formula of course cannot cover the whole curve-length for arbitrary curve, but it has to provide high precision for long length segment.

An input for the arc-length formula must be consistent with the interpolating task. When we interpolate any curve, we investigate and adjust the segments on this curve, using both derivatives at the segment-ends, which are available from evaluator. Same derivatives can be used at the input for the arc-length computation. It brings us to the pure mathematical task: we have points, 1-st and 2-nd derivatives at both ends of segment - we need to find the arc-length, which will simultaneously be a parameter-length for the self-consistent D5-polynomial representation.

To solve this problem we consider the arc-length integral:

$$L = \int_0^1 dt * [\, (dX/dt)^2 + (dY/dt)^2 + (dZ/dt)^2 \,]^{1/2}, \qquad \text{where X, Y, Z are given by}$$

Hermit - representations: $X = X_0 * H_{05}(t) + X_0' * L * H_{15}(t) + \ldots$ (see above).

Here we have non-linear task, because the arc-length L depends on itself. We decompose the irrational expression and use the disturbance theory. The result of such tedious procedure is a compact direct formula:

$$L/|\Delta\mathbf{P}| = 1 + |\mathbf{a}|^2/6 + |\mathbf{a}|^4/280 - |\mathbf{a}|^2*(\mathbf{b}\cdot\boldsymbol{\rho})*2/21 + |\mathbf{a}|^2*(\mathbf{c}\cdot\boldsymbol{\rho})/210 - (\mathbf{a}\cdot\mathbf{d})/30$$
$$+ |[\mathbf{b}\star\boldsymbol{\rho}]|*3/14 + |[\mathbf{c}\star\boldsymbol{\rho}]|/1260 - ([\mathbf{b}\star\boldsymbol{\rho}]\cdot[\mathbf{c}\star\boldsymbol{\rho}])/42;$$

There are some small parameters, which form the solution: ($\mathbf{P}^s$ and $\mathbf{P}^e$ - start & end points)

$$\Delta\mathbf{P} = \mathbf{P}^e - \mathbf{P}^s, \quad \boldsymbol{\rho} = \Delta\mathbf{P}/|\Delta\mathbf{P}|; \quad \mathbf{a} = \tfrac{1}{2}*(d\mathbf{P}^e/dl - d\mathbf{P}^s/dl), \quad \mathbf{b} = \tfrac{1}{2}*(d\mathbf{P}^s/dl + d\mathbf{P}^e/dl) - \boldsymbol{\rho},$$

$$\mathbf{c} = \tfrac{1}{2}*|\Delta\mathbf{P}|*(d^2\mathbf{P}^e/dl^2 - d^2\mathbf{P}^s/dl^2); \qquad \mathbf{d} = \tfrac{1}{2}*|\Delta\mathbf{P}|*(d^2\mathbf{P}^s/dl^2 + d^2\mathbf{P}^e/dl^2) - 2*\mathbf{a};$$

Each of these parameters show the deviation from actual curve-segment to its polynomial representation. Here the derivatives by the arc-length can be easily represent in terms of derivatives by any parameters, so we can obtain them from curve-evaluator.

Such representation allows us to keep arc-length tolerance $\sim 10^{-6} \div 10^{-8}$ for the segment with length ~ 1 and curvature ~ 1. It is incomparably more precise then conventional and modern algorithms [2]. Also it works much faster and reliable because it does not need iterative process - it is the *direct* formula.

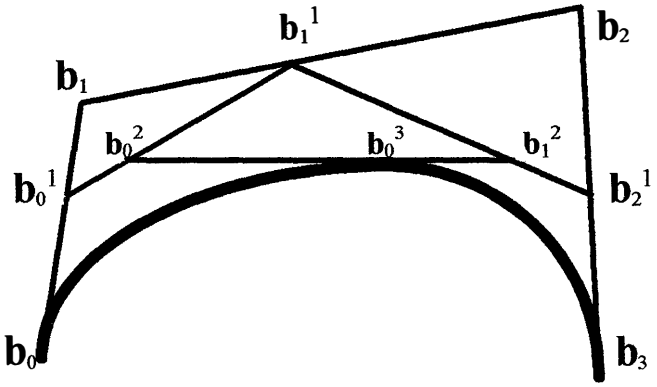
If for any reason we look for the derivatives and arc-length simultaneously - it causes an additional iterative process, which converges very fast in most cases, what was proven by many numerical experiments.

==========

Now we will discuss how to apply all this design, which is based on Hermit splines, to the conventional CAD environment for curves.

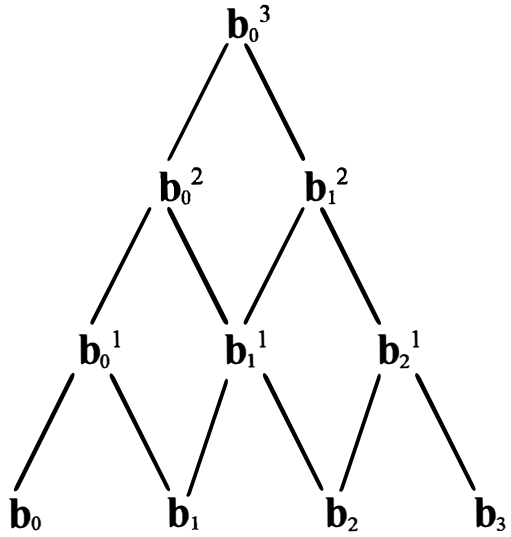
Spline-Curve Converters between Hermit, Bezier, B-spline and Power-basis representations.

Hermit approach is definitely most convenient for interpolating and fitting. However, for historical reasons - the CAD software, accumulated for the last 25 - 30 years was based on NURBs. This is definitely NOT optimal for the performance, it provides tedious work for developers, but NURBs became a basis. For the modern developments within CAD industry, after Hermit - representation is reached, the last stage is a conversion to Bezier or B-spline basis. We will describe Bezier approach briefly for the simplest D3 case [1].



$$b_i^{j+1} = b_i^j + t * (b_{i+1}^j - b_i^j),$$

$$t = (u - u_n) / (u_{n+1} - u_n);$$



Geometrical and pyramid structures of D3 - Bezier representation.

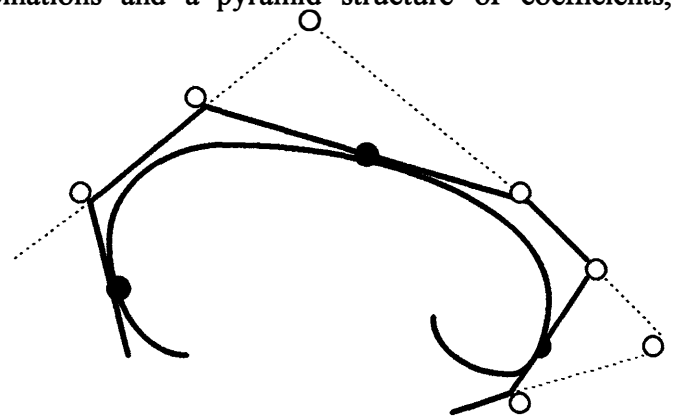
Here instead of 2 end-points and 2 end-derivatives we use 4 Bezier-points as a basis. Now instead of using the direct polynomials - we have step-by-step linear combinations. We see geometrically how it works, and structurally there is a pyramid of coefficients.

However if we were to combine back all the terms for D3-polynomials and represent a result through the initial set of Bezier-points, we come to a Bezier - decomposition:

$$\mathbf{P}(u) = \sum \mathbf{b}_m * B_m^3(t); \qquad B_m^n(t) = t^m * (1-t)^{n-m} * n!/[m! * (n-m)!]$$

Here $B_{mn}(t)$ - Bernstain polynomials. This is a conventional Bezier - representation for CAD environment. When we need to convert spline - segment from Hermit to Bezier basis, we represent Hermit polynomials $H_{mn}(t)$ through Bernstain polynomials $B_{mn}(t)$ and substitute it to the spline $\mathbf{P}(u) = \mathbf{P}_n * H_{03}(t) + \Delta_n * \mathbf{P}_n' * H_{13}(t) + \ldots$, which we discussed before. Then we pick out $B_{mn}(t)$ and combine coefficients at them, which will be the Bezier-points $\mathbf{b}_m$ we are looking for: $\mathbf{b}_m = \Sigma\ (\alpha_{mn} * \mathbf{P}_n + \beta_{mn} * \mathbf{P}_n')$.

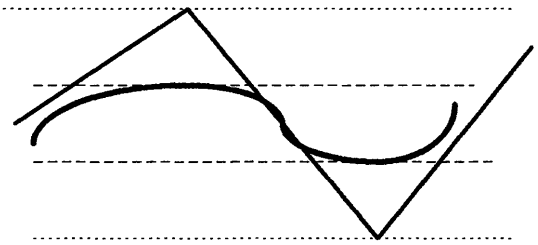
Unfortunately, this is not last step to join CAD - environment, because Bezier basis is only the preliminary basis for B-spline representation. This approach is very close to Bezier - it uses step-by-step linear combinations and a pyramid structure of coefficients, but instead of Bezier-points it uses another basis, called Control - Points. This representation allows to reduce the amount of data, if the derivatives are continuous. For example: in case of 2-nd derivatives continuity we can make only ONE new point instead of THREE for each segment. Many years ago, when memory was expensive, such an approach looked beneficial, but step-by-step it was converted to the whole science: many properties, peculiarities, complicated algorithms, knots insertion procedure, degree elevation procedure and so on. Unfortunately, such an approach became a conventional basis for CAD industry [1, 4, 5]. If we are going to put any modern development as a plug in module to CAD system, we have to provide a converter from Hermit to B-spline representation.



Example of conversion from Bezier-Points to Control-Points

The fastest way for spline-evaluation is, definitely, a direct Power basis computation for curves and surfaces, using Gorner's scheme. For D3 and D5 environment this Power basis is stable and has many advantages, compared with other representations. For example, many applications require very fast computation of the important properties, like perpendicular from given point to curve or surface, mass-properties, etc. When we do it numerically, using the iterative processes, we immediately get the stability and performance problems. The direct Power representation allows us to do it analytically and to forget about all these troubles.

Similar situation repeats for a boundary box computation, which is extremely important to solve intersection problems. The Control-Points from B-spline representation give the simplest restrictions for the curve or surface, according to B-spline's properties [1,2]. However a boundary box, based on these Control-Points is usually very large and for this reason it is not optimal for intersection algorithms. A very narrow boundary box can be computed analytically from Power basis.



Examples of boundary boxes

Now let's discuss the **Spline - Surface** modeling field.

We have the same set of problems for *Surfaces*, which we had for curves:

- Analysis of evaluation technique for the single u-v patch
- Conversion from Hermit basis to Bezier, B-spline and Power basis
- Search of all the partials from the information available (usually from points)
- Correct re-parametrization for special cases

We will consider some of aspects for each step.

For a single u-v patch we have several parametrical representations:

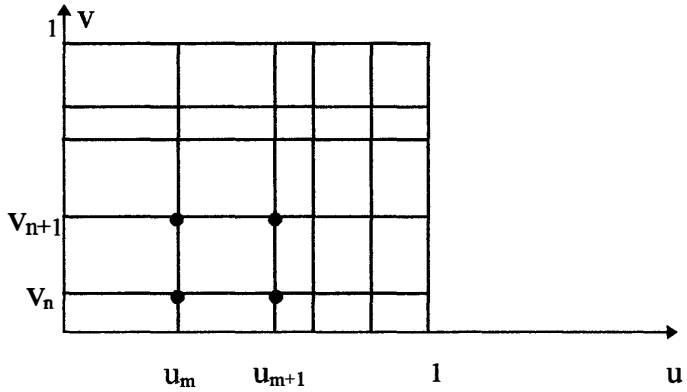
$$\mathbf{P}(u, v) = \sum \mathbf{c}_{mn} * t_u^{\,m} * t_v^{\,n} =$$

$$= \sum \mathbf{h}_{mn} * H_m(t_u) * H_n(t_v) =$$

$$= \sum \mathbf{b}_{mn} * B_m(t_u) * B_n(t_v);$$

$$t_u = (u - u_m) / (u_{m+1} - u_m);$$

$$t_v = (v - v_n) / (v_{n+1} - v_n);$$



Example of u-v parametrical space

In the case of the Hermit representation, each of vector-coefficients $\mathbf{h}_{mn}$ is proportional to the points or partial derivatives. For Bezier, B-spline and other representations the correspondent coefficients can be composed by these partials. Here every knot must have a set of known values: ($\mathbf{P}$ $\mathbf{P}_u$ $\mathbf{P}_v$ $\mathbf{P}_{uv}$) - for 3x3 Degrees;
($\mathbf{P}$ $\mathbf{P}_u$ $\mathbf{P}_v$ $\mathbf{P}_{uv}$ $\mathbf{P}_{uu}$ $\mathbf{P}_{vv}$ $\mathbf{P}_{uuv}$ $\mathbf{P}_{uvv}$ $\mathbf{P}_{uuvv}$) - for 5x5 Degrees.

One can ask - 'why are there so many approaches ?' The point is, all these representations are necessary for different purposes:

- Hermit representation allows to find the partials
- Bezier and B-spline representations allow to switch to CAD environment [1].
- Power - basis allows to make the fastest evaluation, using Gorner scheme

The conversion between all surface - representations is a very tedious procedure. For example, simple 3 x 3 polynomials require 4 x 4 = 16 vector - coefficients for each patch. It means that for conversion to another basis we have to solve 16 linear equations. For quintic case (5 x 5 degrees) we will have 36 equations. Fortunately there are many zeros. This helps a lot. All these conversions were computed for the most popular situations: cubic and quintic. Now we have direct representations for the coefficients $\mathbf{b}_{mn}$ and $\mathbf{c}_{mn}$ through the Hermit coefficients $\mathbf{h}_{mn}$. All these matrixes were tested analytically and numerically. Their usage for direct surface evaluation allows to avoid the general matrix inversion, which is extremely time consuming process.

Also this exercise gave much more than only analytical coefficients for conversion from one basis to another. It allows to prove analytically an extremely important result: D3-representation cannot provide C2-continuity for *surfaces* (curvatures will be broken on the iso-lines: $\mathbf{P}_{uu}$ will be discontinuous in u-direction while we pass through iso-lines u= const; and $\mathbf{P}_{vv}$ will be discontinuous in v-direction while we pass through iso-lines v= const). This fact was the major reason for us to start D5 - environment elaboration, as we mentioned before.

Next we can discuss the computational algorithm to find partial derivatives which are necessary to fulfill the Hermit representation. Usually we have only points, but NOT derivatives, and we have to find the unknown partials from these points for each of the u-v knots. For this purpose we can divide this procedure by several steps. First we run through each iso-curve v = const to find u-derivatives $\mathbf{P}_u$. Here we solve absolutely the same task, which we discussed before for curve environment: it is the 3 - diagonal matrix inversion. However, these iso-curves have unified parametrization, so for all of them this matrix will be the same, and we can make such matrix inversion only once. Then we run in another direction u = const to find v-derivatives $\mathbf{P}_v$ and twists $\mathbf{P}_{uv}$. Here again we use only one 3 - diagonal matrix for all iso-curves and make inversion once, so we do not waste extra time - we compute $\mathbf{P}_u$ and $\mathbf{P}_{uv}$ for all iso-curves, using the same inverse coefficients. It bring us to the very important statement. The discussed methodology provides the highest performance, and the derivatives' computation time for each point practically does not depend on the amount of points. Indeed, such algorithm requires minimum amount of computer operations - this amount is proportional to 'u' and 'v' dimensions.

To compare with conventional tensor - product inversion algorithm we just remind that its performance is proportional to the <u>cube</u> of dimensions. Indeed, the tensor - product interpolation procedure uses a parametrical decomposition in u-v directions [1]:

$\mathbf{P}(u, v) = \sum\sum \mathbf{d}_{mn} * A_m(u) * B_n(v)$ with known net-points $\mathbf{P}(u_m, v_n)$, known basis - functions A(u), B(v) and unknown coefficients $\mathbf{d}_{mn}$. The inversion task requires to invert both matrixes: for 'u' and 'v' directions. Such inversion for general cases takes a huge amount of operations, being proportional to the <u>cube</u> of dimensions.

Now we will discuss one of the basic methods to create the surface.

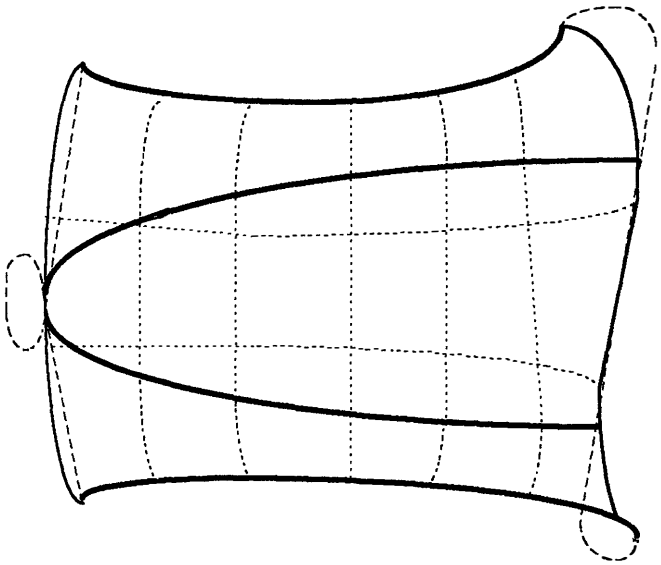
Lofting procedures.

This is one of the main kinds of the non-analytical CAD - surfaces, among *blending, sweeping, offsetting & mesh surfaces*. Loft surface is constructed from the set of input - curves. They can be represented as *curves*, having evaluators, or as *arrays of points*. The goal is to prepare a parametrical spline - surface, passing through input - curves with the tolerance specified and being as smooth as possible.

A quality of the loft-surface depends on two aspects: 1) on the quality of the input - curves and 2) on construction methodology. First we will discuss the conventional CAD approach to solve this problem and see - why it is bad. Then we will discuss the better approach.

Iso-curves lofting approach problems

The conventional way to make a loft-surface considers the input curves as iso-curves in u-direction at the parametrical u - v space. It means that all the points at each input-curve have the same v-parameter. This will not cause problems, if input-curves are slightly non-parallel to each other. However, if they are very non - parallel, it disturbs new curves in v-direction and increases their curvatures up to the unpredictable false loops appearance. Such a situation is a death for manufacturing, if they use such spline - representation.



Example of false loops appearance when wrong lofting method is used

To understand this phenomena we have to remember the re-parametrization problem for curves. Here we have exactly the same situation, which we discussed before: two points at the neighbouring input-curves are physically close to each other, but their v-parameters are quite different, so the parameter-length between them is unnaturally big. We can make this parameter small but it will immediately cause troubles at the other side of this example, where the natural v-parameter distance is big enough. This situation is a result of wrong iso-curve re-parametrization approach.

One can try non-uniform parametrization with v-knots proportional to the average distance between the curves. It gives better results but still does not solve the problem.

The only way to avoid these troubles is to deny the bad conventional concept about input-curves to be iso-curves. That is why a conceptually new approach, the *loft re-parametrization algorithm* was developed. The idea is simple, but nothing is for free, so the new way was not easy. For example, while the 1-st version was developed, the algorithm was unstable in u-direction. Instead of the 1-st monster in v-direction (false loops) we got the 2-nd monster - oscillations in u-direction. The surfaces looked very funny. However, there was a suspicion, that between these two beasts has to exist a beauty - a surface, smooth enough in both directions. Such algorithm was found and we will discuss it now.

We suppose that each point at each input-curve has its own v-parameter, so we deny the input-curves to be iso-curves. Only boundary curves: the 1-st & the last are iso-curves: the 1-st curve corresponds to $v = 0$, the last one - to $v = 1$. The stable loft-surface constructional algorithm consists of 7 steps:

1. We make the optimized input-curves from original set of curves or points.
2. We make the unified u-parametrization simultaneously for all input-curves.
3. We take the points from all input-curves at each u-parameter and construct v-curves in v-direction, providing *natural* v-parametrization.
4. We make the unified v-parametrization simultaneously for all v-curves.
5. We pass through all iso-lines ($v = const$) to find u-derivatives.
6. We pass through all iso-lines ($u = const$) to find v-derivatives and twists.
7. We convert all these derivatives from Hermit basis to Bezier or B-spline representation depending on the user's needs.

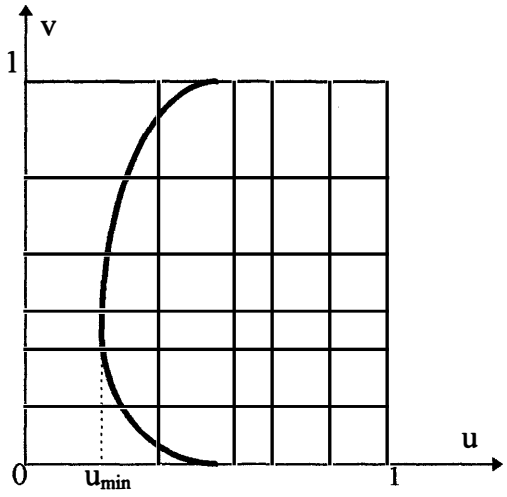
That's it. The result is - a smooth loft-surface, constructed from the very non-parallel set of input-curves, which gave false loops within conventional CAD algorithm. Now the new loft-surface is smooth in v-direction, because v-parametrization is natural.

Also we have to add that each step of this algorithm takes care about tolerance specified and the loft-surface passes through the original set of curves. The procedure also tests automatically for the surface being open or periodical in 'u' or 'v' directions, and if there are any kinds of singularity - to mark the surface properly.

One more thing we have to mention here. The direct spline construction is not the only way for loft-surface creation. We can make procedural loft-surface, using a similar, but much easier algorithm. However, after that we need to convert this procedural surface to the optimized spline-surface. It is not easy, but this task is very important for many kinds of procedural surfaces and it deserves a specific consideration, which we will discuss now.

Universal converter from any parametrical surface to the optimized Spline

We already discussed such procedure for curves. Now we will talk about its generalization for surfaces. Let's consider u-v parameter space and investigate the 1-st iso-line: $v = 0$. We can start from a very small segment, make polynomial representation and test, if it fits the iso-curve segment. If yes we expand the probing segment. If no - we cut it twice. Then we make iterations to find the biggest possible 'good-segment'. This is absolutely the same part of analytical - numerical fitting procedure, which we discussed before for the curves' environment. Here we mark such furthest 'good u-point' and repeat the same procedure with many iso-curves in u-direction. Then we make a specific curve, passing through all such 'good u-points' and find its minimum 'u_{min}'.



Good-u-curve in u-v space

We can state that all the iso-curves in u-direction will fit the polynomial representation at the segment (0 -> u_{min}). Now u_{min} becomes our new start-point (instead of 0) and we repeat this process until the end of u-parameters. Then we make the same procedure in v-direction. As a result we have a new u-v grid.

Now we can find all the points and partial derivatives from the original surface at the new parameter - knots. It gives us the Hermit representation. After this we make a conversion from Hermit-basis to Bezier or B-spline basis. A new surface is ready: it is a spline-surface; it fits the original surface; and it has the minimum amount of u-v patches.

Now we will discuss the most exiting problem.

Local Surface Deformations

In recent years, deformation procedures for curves and surfaces have become more popular for CAD and 3D games developers. Old methods, accumulated for many years in B-spline technique, allowed to modify surfaces using the Control-Points manipulation, which effect surface directly, but non-controllable. To adjust a large part of the surface, the user had to change many Control - Points simultaneously and very consistently to avoid unexpected surface changes. The user had to improvise to do this, which requires a highly qualified user, a lot of time and money. More and more, users have been asking for the automation of this process, looking for a more natural way to handle surface deformations.

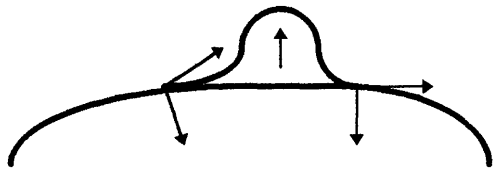
The simplest way to solve this problem is to use a good surface-interpolator, moving through-points, which lie on the surface, as opposed to moving Control-Points. However it covers only part of the problem, because the surface-interpolator changes the *whole* surface while for many cases we need to adjust only some region.

Next step is to isolate a set of points or curves on the surface and to adjust the whole surface except this set. User can make deformations by blowing up the surface, as a balloon, or using other methods. However, following this way, we break the smoothness of the surface, so this method could be useful only for the very specific cases.

In the conventional situation user does NOT need to touch the 'good' part of the surface, but has to adjust the 'bad' part, keeping smooth contact between the two. This problem arises usually after the user had constructed any kind of surface (blending, lofting, sweeping, offsetting, etc.) and was satisfied with some 'good' part of the surface, but another 'bad' part did not meet the requirements or expectations.

We consider a two-stage tool to cover such problems. The first tool will help user to restrict the 'bad' part of the object; the second tool will help to adjust this 'bad' part.

For local *curve* deformation, these tools seem very natural. The user only clicks two points on the curve with a mouse (1-st stage); then moves the mouse (2-nd stage), changing curvatures or simulating pressure to deform the curve segment between these end-points.



Local curve deformation.

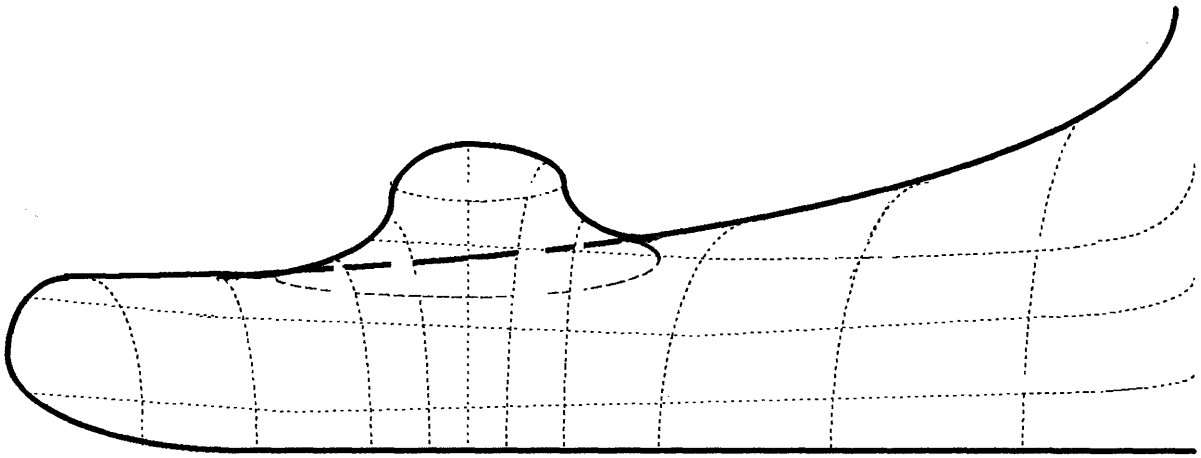
Here we keep the derivatives at the ends to provide C2-continuity, and do NOT allow changes to move outside end-points. Here we can see the advantage compared with the B-spline approach, because any move of a Control-Point will cause *non*-controllable changes of the curve (sometimes - waves). These changes propagate for the distance of (Degree + 1) knots and we cannot stop them.

We also can use the new approach for drawing curve by hand. If our hand - writing is not smooth enough, we can use this tool to smooth any part of curve automatically.

Generalizing about this natural way for modifying surfaces, the following procedure would be used:

1. For the 1-st stage, the user would click some points or draw a curve around the 'bad' part of the surface. This 1-st tool would automatically prepare the corresponding transitional spline-curve with its normal and curvatures, which will be in use at the 2-nd stage as boundary conditions between 'good' and 'bad' parts.

2. At the 2-nd stage, the user would move the mouse around the 'bad' part of the surface, deforming it. However for each move, the 'good' part and deformed part will automatically have smooth contact with each other, based on everything being prepared preliminary at the 1-st stage.



Such tools can be either a plug-in module or stand alone program, even a computer game for children, because its interface does not require B-spline's knowledge. It can be very useful for many applications, such as deformations of human faces, bodies, cartoon development, animation, design of clothing and shoes, etc. In a CAD environment this software can be used as a powerful design tool. A designer can separate a surface from solid body file, for example, an automobile construction. Then he can make smooth changes and reinsert this surface back into the original file with the same boundaries, tangents and curvatures.

This product covers the local deformation problem naturally, quickly and accurately. Such things become possible employing the new approaches, presented in this paper.

REFERENCES

1. F. Farin, *Curves and Surfaces for Computer Aided Geometric Design*. 3-rd. edition, Academic Press, New York, 1993.

2. R.P. Markot and R.L. Magedson, "Solutions of Tangential Surface and Curve Intersections", *Computer Aided Design,* Vol. 21, No.7, 1989, pp.421 - 429.

3. F-C Wang and D C H Yang, "Nearly arc-length parametrized quintic-spline interpolation for precision machining", *Computer Aided Design,* Vol. 25, No.5, 1993, pp.281 - 288.

4. W. Boehm, "Inserting new knots into B-spline curves", *Computer Aided Design,* 12, 1972, pp.199 - 201.

5. STEP "Standard for Exchange of Product Model Data", ISO 10303.

CENTER FOR COMPUTATIONAL MATHEMATICS REPORTS

University of Colorado at Denver
P.O. Box 173364, Campus Box 170
Denver, CO 80217-3364

Fax: (303) 556-8550
Phone: (303) 556-8442
http://www-math.cudenver.edu/

81. F. Brezzi, L.P. Franca, T.J.R. Hughes and A. Russo, "$b = \int g$."
82. L.P. Franca, C. Farhat, M. Lesoinne and A. Russo, "Unusual Stabilized Finite Element Methods and Residual-Free-Bubbles."
83. F. Brezzi, L.P. Franca, T.J.R. Hughes and A. Russo, "Stabilization Techniques and Subgrid Scales Capturing."
84. J. Mandel, R. Tezaur and C. Farhat, "A Scalable Substructuring Method by Lagrange Multipliers for Plate Bending Problems."
85. K. Kafadar, P.C. Prorok and P.J. Smith, "An Estimate of the Variance of Estimators for Lead Time and Screening Benefit in Randomized Cancer Screening Trials."
86. H.J. Greenberg, "Rim Sensitivity Analysis from an Interior Solution."
87. K. Kafadar, "Two-Dimensional Smoothing: Procedures and Applications to Engineering Data."
88. L.P. Franca, C. Farhat, A.P. Macedo and M. Lesoinne, "Residual-Free Bubbles for the Helmholtz Equation."
89. Z. Cai, J.E. Jones, S.F. McCormick and T.F. Russell, "Control-Volume Mixed Finite Element Methods."
90. D.C. Fisher, K. Fraughnaugh and S.M. Seager, "Domination of Graphs with Maximum Degree Three."
91. L.S. Bennethum, M.A. Murad and J.H. Cushman, "Clarifying Mixture Theory and the Macroscale Chemical Potential."
92. L.S. Bennethum and T. Giorgi, "Generalized Forchheimer Equation for Two-Phase Flow Based on Hybrid Mixture Theory."
93. L.P. Franca and A. Russo, "Approximation of the Stokes Problem by Residual-Free Macro Bubbles."
94. H.J. Greenberg, A.G. Holder, C. Roos and T. Terlaky, "On the Dimension of the Set of Rim Perturbations for Optimal Partition Invariance."
95. W.A. Lodwick and K.D. Jamison, "Interval Methods and Fuzzy Optimization."
96. S.C. Billups, "Improving the Robustness of Complementarity Solvers Using Proximal Perturbations."