

Genetic B-Spline approximation on combined B-reps

Matthias Bein · Dieter W. Fellner · André Stork

Published online: 19 April 2011
© Springer-Verlag 2011

Abstract We present a genetic algorithm for approximating densely sampled curves with uniform cubic B-Splines suitable for Combined B-reps. A feature of this representation is altering the continuity property of the B-Spline at any knot, allowing to combine freeform curves and polygonal parts within one representation. Naturally there is a trade-off between different approximation properties like accuracy and the number of control points needed. Our algorithm creates very accurate B-Splines with few control points, as shown in Fig. 1. Since the approximation problem is highly nonlinear, we approach it with genetic methods, leading to better results compared to classical gradient based methods. Parallelization and adapted evolution strategies are used to create results very fast.

Keywords Spline · Approximation · Genetic · Parallel · Combined B-reps · Subdivision

1 Introduction

B-Spline representations are commonly used in CAD. Example application areas are freeform modeling, reverse engineering, animation, simulation and visualization. Often, only measurements are available at the beginning, for instance, created by scanning or inserted by sketching meth-

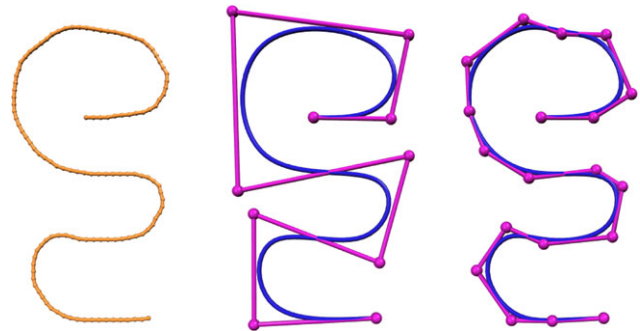


Fig. 1 Given a densely sampled input curve (*left*), our algorithm automatically extracts a few characteristic B-Spline control points (*middle*) instead of a large set of imprudent control points (*right*)

ods, like illustrated in Fig. 2. In this case a B-Spline has to be fitted to the given input points by approximation.

The process of fitting a B-Spline to measurements is a complicated nonlinear optimization problem. Generally, several parameters have to be determined in order to execute a specific approximation: the degree of the base functions, the parameterization of the input points, the number of control points and the knot vector. The approximation itself is a optimization problem too, in which a predefined error function gets minimized. Even in our specialized case of uniform cubic B-Splines with marked knots, there are many unknowns to be determined, most importantly the parameterization of the input curve.

1.1 Related work

De Boor [3], Farin [6] and Cohen [5] provide considerable detail about curve representations. An introduction to the problem of interpolation and approximation with B-Splines is given by Shene [16]. Optimal parameterization methods have been investigated in Hoschek [11] and Speer [17], but

M. Bein (✉) · D.W. Fellner · A. Stork
TU Darmstadt & Fraunhofer IGD, Fraunhoferstrasse 5,
64283 Darmstadt, Germany
e-mail: matthias.bein@gris.tu-darmstadt.de

D.W. Fellner
e-mail: d.fellner@igd.fraunhofer.de

A. Stork
e-mail: andre.stork@igd.fraunhofer.de

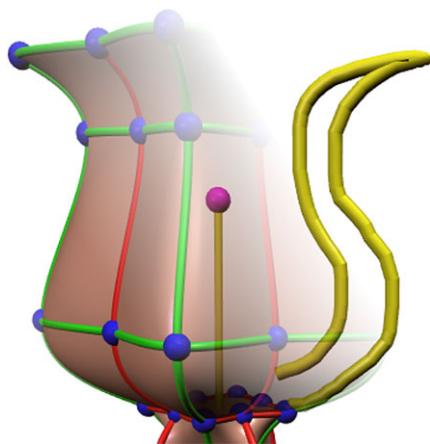


Fig. 2 Example application of a sketch based system. A stroke, indicating a rotational extrusion, is approximated with a B-Spline. Its control polygon is used to calculate the control mesh for a subdivision surface. A characteristic control polygon is important for further modeling operations

cannot find a global minimum over all possible approximations. The knot vector has been treated as a parameter in Sapidis [15] and Juhasz [12]. A global optimization problem, which treats the parameterization and knot vector at once, is given by Gengoux [13]. However, these methods are not applicable to our uniform representation.

An overview of genetic algorithms in CAD is given by Renner [14]. Markus introduces genetic interpolation in [1] and [2], limited to Bezier curves and Hermite splines. Genetic B-Spline approximation is examined by Goldenthal in [7], applied to non-uniform B-Splines. In summary, existing methods explicitly use a different representation, use the knot vector as parameter, do not scan the solution space for a global minimum or require a lot of time to generate results.

A main motivation for us to choose uniform cubic B-Splines is its application with *Combined B-Reps*, presented by Havemann [9, 10]. Combined B-Reps is a surface representation using an extended Catmull/Clark [4] subdivision scheme. The subdivision rules originate from bicubic uniform B-Splines and are enhanced with smooth and sharp edges which determine the continuity property of the surface, hence our B-Spline representation.

1.2 Outline

This work focuses on densely sampled input curves. In contrast to sparse input points there is hardly a doubt of where the curve surface is. Our three main contributions are:

1. Great reduction of complexity by minimizing the number of control points needed to approximate the original shape. This leads to control polygons with few, but characteristic control points.

2. Accurate B-Splines with a small error to the input curve and the possibility to control accuracy and complexity.
3. Calculation within short time, suitable for interactive applications. Parallelization, heuristics and adapted strategies should be used in order to cope with the huge solution space of the approximation problem.

The remaining of the paper is structured as follows: In Sect. 2, we describe the pipeline we use for approximating a densely sampled input curve with a B-Spline. In Sect. 3, we present the genetic framework with its evolution strategies. Results of our work are presented and analyzed in Sect. 4. Finally, in Sect. 5, we discuss limitations and possible improvements.

2 B-Splines

In this section, we describe the considered B-Spline and present a pipeline for fitting a B-Spline to an input curve, which is defined by a densely sampled set of points. Further, we motivate the representation of our genes used in the genetic algorithm.

2.1 B-Spline definition

A B-Spline $C(t)$ is defined by n control points $P = \{\mathbf{p}_0, \dots, \mathbf{p}_{n-1}\}$ and base functions $N_{i,k}(t)$ of degree k :

$$C(t) = \sum_{i=0}^{n-1} N_{i,k}(t) \cdot \mathbf{p}_i. \tag{1}$$

The basis functions are defined recursively over the knot vector $U = \{u_0, u_1, \dots, u_{m-1}\}$, a non-descending sequence of $m = n + k + 1$ scalar values, with

$$N_{i,0}(t) = \begin{cases} 1 & \text{for } u_i \leq t \leq u_{i+1}, \\ 0 & \text{otherwise,} \end{cases} \quad \text{and} \tag{2}$$

$$N_{i,k}(t) = \frac{t - u_i}{u_{i+k} - u_i} N_{i,k-1}(t) + \frac{u_{i+k+1} - t}{u_{i+k+1} - u_{i+1}} N_{i+1,k-1}(t). \tag{3}$$

In this work, we focus on uniform cubic B-Splines, which implies the conditions

$$k = 3 \quad \text{and} \tag{4}$$

$$u_{i+1} - u_i = c \quad \text{for } i = \{0, \dots, m - 1\}. \tag{5}$$

In particular, we use $c = 1$, which reduces the recursive definition of the base functions to fixed polynomial functions.

As mentioned in the introduction, knots may be marked sharp, which forces the B-Spline to interpolate the corre-

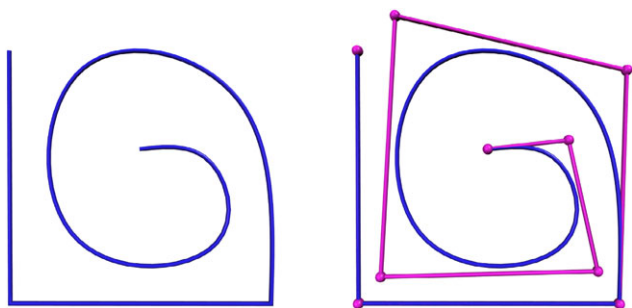


Fig. 3 Example B-Spline: (left) B-Spline curve; (right) overlay with its control polygon. Starting from the top left, the first three and the last knots are marked sharp. The corresponding control points are interpolated

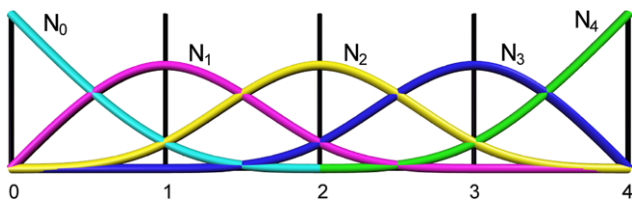


Fig. 4 All five basis functions within the clamped interval [0, 4]. The knot vector $U = \{0, 1, 2, 3, 4\}$ is uniform and does not contain multiple knots

sponding control point with C^0 -continuity, as illustrated in Fig. 3. This can be achieved by setting the multiplicity of the knot to k , but that violates the condition of uniform knots. Therefore, we use slightly different base functions which can be clamped at any knot. The first three basis functions, clamped at $t = 0$, are

$$N_0(t) = \begin{cases} \frac{1}{6}t^3 - t + 1, & t \in [0, 1], \\ \frac{1}{6}(2 - t)^3, & t \in [1, 2], \end{cases} \tag{6}$$

$$N_1(t) = \begin{cases} -\frac{1}{3}t^3 + t, & t \in [0, 1], \\ \frac{1}{6}(3t^3 - 15t^2 + 21t - 5), & t \in [1, 2], \\ \frac{1}{6}(3 - t)^3, & t \in [2, 3], \end{cases} \tag{7}$$

$$N_2(t) = \begin{cases} \frac{1}{6}t^3, & t \in [0, 1], \\ \frac{1}{6}(-3t^3 + 12t^2 - 12t + 4), & t \in [1, 2], \\ \frac{1}{6}(3t^3 - 24t^2 + 60t - 44), & t \in [2, 3], \\ \frac{1}{6}(4 - t)^3, & t \in [3, 4]. \end{cases} \tag{8}$$

The other basis functions $N_i(t)$ with $i > 2$ can be evaluated with $N_2(t - i + 2)$. If the basis functions are clamped at a parameter a instead of 0, they can be evaluated with $N_{i-a}(t - a)$. Symmetry allows us to evaluate a basis function which is clamped at a parameter $b > t$ as shown in Fig. 4. Note that with this representation the number of knots in U is the number of control points in P , hence $n = m$. These basis functions reflect the subdivision scheme used in Combined B-Reps.

2.2 B-Spline approximation

Given the task to fit the above described B-Spline to an input curve defined by l points $D = \{\mathbf{d}_0, \dots, \mathbf{d}_{l-1}\}$, the following system of equations can be build:

$$\mathbf{d}_j = C(t_j) = \sum_{i=0}^{n-1} N_i(t_j) \cdot \mathbf{p}_i, \tag{9}$$

$$\underbrace{\begin{pmatrix} \mathbf{d}_0 \\ \vdots \\ \mathbf{d}_{l-1} \end{pmatrix}}_D = \underbrace{\begin{pmatrix} N_0(t_0) & \dots & N_{n-1}(t_0) \\ \vdots & \ddots & \vdots \\ N_0(t_{l-1}) & \dots & N_{n-1}(t_{l-1}) \end{pmatrix}}_N \underbrace{\begin{pmatrix} \mathbf{p}_0 \\ \vdots \\ \mathbf{p}_{n-1} \end{pmatrix}}_P. \tag{10}$$

For $l \gg n$, this system is over-determined and has possibly no solution. From linear algebra, we know that a unique least squares solution exists and can be found by performing the following conversion:

$$D = N \cdot P, \tag{11}$$

$$\underbrace{N^T \cdot D}_Q = \underbrace{N^T \cdot N}_M \cdot P, \tag{12}$$

$$Q = M \cdot P. \tag{13}$$

Solving $Q = M \cdot P$ with the unknown control points P leads to the solution of $D = N \cdot P$ in the least squares sense. By construction, we know that M is a square matrix of size $n \times n$, symmetric and positive definite. Therefore, Cholesky Decomposition [8] can be used. In the case of interpolating the endpoints, the systems size can be reduced by 2 rows and columns, leading to the final size of $(n - 2) \times (n - 2)$.

The error ϵ which gets minimized by this system is defined by the summed squared distances between the input points \mathbf{d}_j and points on the B-Spline C evaluated at parameter t_j :

$$\epsilon = \sum_{j=0}^{l-1} \|\mathbf{d}_j - C(t_j)\|^2. \tag{14}$$

2.3 Assigning parameters

In summary, approximating the B-Spline to a discrete input curve comes down to specifying parameters t_j for each input point \mathbf{d}_j . The process of determining parameters is not an easy task. There are unlimited possible parameterizations which greatly effect the shape of the approximated B-Spline. Depending on the features of the input curve, the parameters have to be chosen wisely in order to allow the approximated B-Spline to capture the original shape of the curve.

Figure 5 illustrates the problem of parameterization. In the left picture, the yellow input curve is parameterized by chord length within the interval [0, 5]. The knot points with

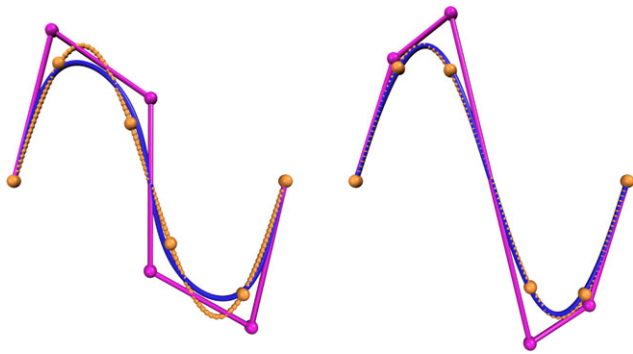


Fig. 5 Influence of different parameterizations: (left) chord length; (right) our algorithm; (yellow) sampled input sine curve; (yellow spheres) knot points; (blue) B-Spline curve; (purple) its control polygon

$t \in \{0, 1, 2, 3, 4, 5\}$ are visualized by yellow spheres. Executing the approximation leads to the blue B-Spline curve with its purple control polygon. It hardly captures the original sine curve. While reparameterization methods may improve the approximation, they may not lead to optimal results, especially in more complex cases. In the right picture, a different placement of the knot points induces a different parameterization of the data points, resulting in a different approximation. This time the shape of the sine curve is well captured with the same number of control points.

In order to assign parameters t_j to the input points \mathbf{d}_j , we construct a mapping function p

$$t_j = p(\text{cl}(\mathbf{d}_j)) \tag{15}$$

which maps the chord length parameterization $\text{cl}(\mathbf{d}_j)$ of a point to its final parameter.

First of all, we calculate the chord length $\text{cl}(\mathbf{d}_j)$ of each input point with

$$\text{cl}(\mathbf{d}_0) = 0, \tag{16}$$

$$\text{cl}(\mathbf{d}_j) = \text{cl}(\mathbf{d}_{j-1}) + \|\mathbf{d}_j - \mathbf{d}_{j-1}\|. \tag{17}$$

Then we define a subset $S = \{\mathbf{s}_0, \dots, \mathbf{s}_{n-1}\} \subset D$. These points reflect the knot points. Therefore, their parameters can be simply assigned by $p(\text{cl}(\mathbf{s}_i)) = i$. Note that the first and the last input points \mathbf{d}_0 and \mathbf{d}_{l-1} are always part of S , since we want the B-Spline to interpolate these points. The number of points in S also defines the number of B-Spline control points.

Since it is natural for a cubic B-Spline to have C^1 -continuous parameterization, we estimate the derivative of the parameterization at every knot point $\mathbf{s}_i \in S$:

$$p'_i = \frac{2}{\text{cl}(\mathbf{s}_{i+1}) - \text{cl}(\mathbf{s}_{i-1})}. \tag{18}$$

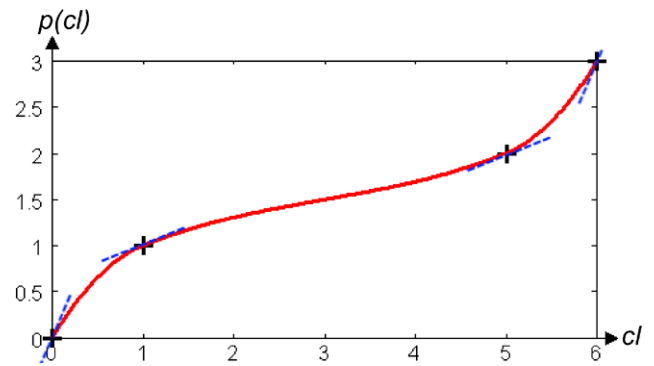


Fig. 6 Mapping function example: (horizontal axis) chord length parameterization $\text{cl} \in [0, 6]$; (vertical axis) final parameter $p \in [0, 3]$. Estimated tangents p' are shown with blue dashed lines

The first and last derivatives are calculated separately:

$$p'_0 = \frac{2}{\text{cl}(\mathbf{s}_1) - \text{cl}(\mathbf{s}_0)} - p'_1, \tag{19}$$

$$p'_{n-1} = \frac{2}{\text{cl}(\mathbf{s}_{n-1}) - \text{cl}(\mathbf{s}_{n-2})} - p'_{n-2}. \tag{20}$$

Now we can construct a cubic polynomial $t_j = p(\text{cl}(\mathbf{d}_j))$ for each segment $[\text{cl}(\mathbf{s}_i), \text{cl}(\mathbf{s}_{i+1})]$ from the following four conditions:

$$p(\text{cl}(\mathbf{s}_i)) = i, \tag{21}$$

$$p(\text{cl}(\mathbf{s}_{i+1})) = i + 1, \tag{22}$$

$$p'(\text{cl}(\mathbf{s}_i)) = p'_i, \tag{23}$$

$$p'(\text{cl}(\mathbf{s}_{i+1})) = p'_{i+1}. \tag{24}$$

Figure 6 shows an example of the mapping function. A curve with chord length parameterization within $[0, 6]$ is mapped to $[0, 3]$. The chord lengths of the knot points $\text{cl}(\mathbf{s}_i) = \{0, 1, 5, 6\}$ are mapped to $p(\text{cl}(\mathbf{s}_i)) = U = \{0, 1, 2, 3\}$, reflecting the uniform knot vector. All points \mathbf{d}_j between two knot points \mathbf{s}_i and \mathbf{s}_{i+1} are mapped to the interval $[i, i + 1]$ with C^1 -continuity to neighboring intervals. Overall this example shows a parameterization suitable for a uniform B-Spline with four control points and knot points \mathbf{s}_i with chord lengths 0, 1, 5 and 6.

Finally, we can calculate a parameter for every input point, evaluate the basis functions, build and solve the system of linear equations of the B-Spline approximation. The whole approximation pipeline has been reduced to specifying the subset S . Choosing a good subset is addressed by the genetic algorithm described in the next section.

In order to include marked knots, each point \mathbf{s}_i of S can be flagged either smooth or sharp. If \mathbf{s}_i is flagged sharp, the basis functions are clamped at the corresponding parameter i . By default, the first and last point is flagged sharp and thus interpolated by the B-Spline. For efficiency, the approximation is separated into parts where only the first and last

point is flagged sharp. This leads to several small systems of linear equations which can be solved faster than a single large system due to the cubic complexity of Cholesky Decomposition.

3 Genetic algorithm

In Sect. 2, we concluded that only two parameters are needed to approximate a B-Spline to an input curve in the least squares sense. The following parameters define a gene for our genetic algorithm:

1. The subset $S \subset D$.
2. The sharpness information $b_i \in B$ for each point $s_i \in S$.

In this section, we analyze the solution space and present our genetic framework and the implemented evolution strategies.

3.1 Solution space

Even with our approximation pipeline, which essentially reduces the degrees of freedom, the space of possible solutions is huge. Every subset $S \subset D$ with its sharpness information leads to a different solution. Given the number l of input points and given that the first and last input points are always part of the subset S , the number of different subsets is 2^{l-2} . For $l = 100$, which is a reasonable size for a densely sampled input curve, there are $2^{98} \approx 317 \times 10^{27}$ subsets. Each subset can vary with different sharpness information too, further increasing the number of possible solutions.

Table 1 illustrates the correlation between the number of input points l , number of free control points n' and the number of approximations (single threaded) per second on an i7 920 CPU. There is a big variance in the number of approximations in dependency of the number of input points and number of control points. Even though the approximation can be easily parallelized for almost linear speedups, the solution space is much larger than the number of approximations possible within a short time. Therefore, compromises have to be done between coarse scanning of the solution space and fast computation time. In this work, we focus on deterministic evolution strategies to create good results within a few approximations.

3.2 Fitness function

In order to compare different solutions among each other, a fitness function f has to be specified. The aim of the genetic algorithm is to find a gene with optimized fitness. As mentioned in the introduction, a small error ϵ is not the only optimization criterion of this work. The number of control points n needed to achieve the small error is important, too.

Table 1 Number of approximations on an i7 920 CPU (single threaded) in dependency of the number of data points (l) and number of free control points (n')

l	$n' = 2$	$n' = 4$	$n' = 6$	$n' = 10$	$n' = 20$
50	98.5k	75.5k	55.7k	33.7k	17.1k
100	63.9k	45.6k	34.6k	20.6k	8.4k
150	47.4k	33.7k	24.6k	14.7k	6.0k
200	37.7k	26.7k	19.4k	11.6k	4.7k
400	20.3k	14.1k	10.3k	4.9k	2.3k

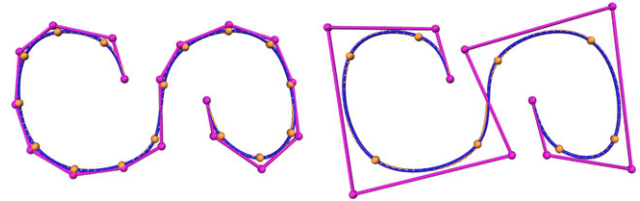


Fig. 7 Consequences of different fitness functions: (left) $f = n \cdot \epsilon$; (right) our exponential function. The simple multiplication tends to generate B-Splines with many control points

Hence the fitness function has to be a combination of both attributes ϵ and n .

The naive approach, a multiplication $f = n \cdot \epsilon$, is not leading to B-Splines with few control points, as illustrated in Fig. 7. This is because the number of control points n has a vanishing influence on the fitness function f , if the error ϵ gets small. With many control points, the approximated B-Spline has a small error automatically. For example, in the case $n = l$ with as many control points as input points, the approximation converts to an interpolation. The error of the interpolation is 0, thus the fitness function is 0, and therefore the best possible solution. Of course, an interpolation with many control points is contrary to our goals.

A main property of the fitness function, whose minimization leads to the desired results, has to have the following behavior: with decreasing error the influence of the number of control points has to rise relatively. We experimented with a simple exponential function

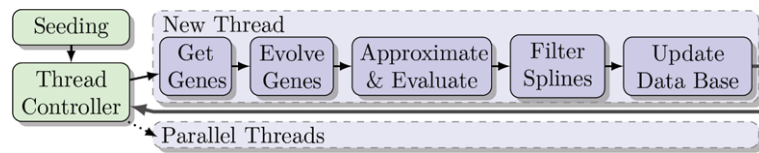
$$f = n \cdot b^\epsilon.$$

With an error close to 0, the exponential part gets close to 1, which leaves the number of control points as the only fitness criterion. With higher error the exponential part grows quickly which relatively reduces the influence of the number of control points.

The base b steers the influence of the error and the number of control points on the fitness function.

A smaller base shifts the influence towards the number of control points n , while a higher base shifts the influence

Fig. 8 Flow graph of our algorithm. The Thread Controller keeps as many threads running as possible



towards the error ϵ . If the base is 1, the error has no influence on the fitness function. A base between 0 and 1 leads to the effect of favoring a large error, which is naturally not recommended.

The approximation detailed in Sect. 2, which minimizes the sum of squared distances, works well together with the above described fitness function. Given a fixed number n of control points, the minimization of the error ϵ within the approximation leads to a minimized fitness function f .

3.3 Pipeline

In previous work, we experimented with iterative approximation methods. The results have been good but not optimal and motivated us to create a framework for genetic B-Spline approximation. It is important to note that every heuristic or iterative algorithm can be integrated in the genetic framework. Therefore, the genetic framework always finds better results but requires more computation time. The mentioned iterative approximation is part of our seeding as well as one evolution strategy.

Figure 8 illustrates the pipeline of our genetic algorithm. It starts with a seeding which generates the first genes. We implemented two different seeding functions:

Uniform: Points $s_i \in S \subset D$ are picked uniformly among all points D for different numbers of control points. The generation of the subset and calculation of the B-Spline can be done in parallel.

Iterative: The iterative approximation method we used in the previous work has been integrated as a seeding function. It starts with two control points only, hence a straight line from the first to the last point of the input curve. In every iteration, the point \mathbf{d}_j with the largest deviation from the B-Spline is added to the subset S . If the point \mathbf{d}_j is close to an existing point $s_i \in S$, s_i is flagged sharp instead of adding \mathbf{d}_j to S . The gene of every iteration is added to the seed population.

After generating the seeds, the evolution begins. A thread controller keeps a certain amount of parallel threads running, usually the number of threads offered by the CPU. Each thread generates and evaluates new genes independently from other threads with the following pipeline:

1. Get a certain number of genes G and remove them from the database.
2. Generate new genes by mutating all genes in G and by recombining the first gene of G , which is the fittest one, with all other genes in G .

3. Approximate and evaluate all B-Splines defined by the new genes.
4. Sort and filter the new B-Splines.
5. Update the data base by merging the remaining B-Splines back into it. Optionally reduce the database size, if there are too many B-Splines, by removing the least fit B-Splines until the desired size is reached.

The next subsections will explain the evolution strategies and filtering in detail.

3.4 Evolution strategies

Evolution strategies can be divided into mutation and combination. The first simply alters a gene on its own. The latter creates a new gene by combining different genes. It is important for the strategies to allow exploration of the whole solution space. Since we focus on real time computation, we also added strategies which deterministically reduce the error of the B-Spline. Keep in mind that every thread executes all evolution strategies independently from other threads. All genes generated this way are collected in a list and filtered subsequently before merging them back into the database. In summary, we implemented five mutation strategies and one combination strategy:

Jiggle: Pick an inner point $s_i \in S$ and replace it by a neighboring point. This is done for every point $s_i \in S$, except for the first and last one, in positive and negative direction. Given m , the number of points in S , a thread creates $2(m - 2)$ new genes. For example,

$$S = \{\mathbf{d}_0, \mathbf{d}_{30}, \mathbf{d}_{70}, \mathbf{d}_{90}\} \xrightarrow{\text{jiggle}+\mathbf{d}_{70}} \{\mathbf{d}_0, \mathbf{d}_{30}, \mathbf{d}_{71}, \mathbf{d}_{90}\} = S'$$

Toggle: Pick an inner point $s_i \in S$ and toggle its sharpness information b_i . $m - 2$ new genes are generated this way. For example,

$$B = \{1, 0, 0, 0, 1\} \xrightarrow{\text{toggle } i=2} \{1, 0, 1, 0, 1\} = B'$$

Delete: Delete a point $s_i \in S$. Executing this operation for all inner points in S leads to $m - 2$ new genes. For example,

$$S = \{\mathbf{d}_0, \mathbf{d}_{30}, \mathbf{d}_{70}, \mathbf{d}_{90}\} \xrightarrow{\text{delete } \mathbf{d}_{70}} \{\mathbf{d}_0, \mathbf{d}_{30}, \mathbf{d}_{90}\} = S'$$

Insert: Pick an interval $[s_i, s_{i+1}]$ and insert a new point. In our implementation, the inserted point has the largest deviation from the B-Spline among all points of the interval, in

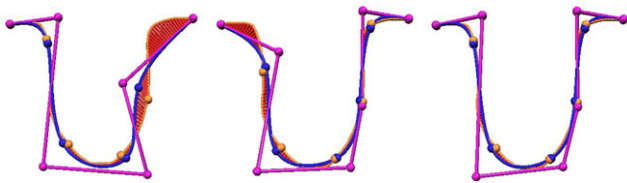


Fig. 9 Recombination: (left) father B-Spline; (middle) mother B-Spline; (right) recombination with the best parts of both B-Splines

order to reduce the error systematically. When done with all intervals, this adds $m - 1$ new genes. For example,

$$S = \{\mathbf{d}_0, \mathbf{d}_{30}, \mathbf{d}_{90}\} \xrightarrow{\text{insert } \mathbf{d}_{70}} \{\mathbf{d}_0, \mathbf{d}_{30}, \mathbf{d}_{70}, \mathbf{d}_{90}\} = S'$$

Merge: Two knot points are merged by averaging them. $m - 3$ inner intervals can be merged to create new genes. For example,

$$S = \{\mathbf{d}_0, \mathbf{d}_{30}, \mathbf{d}_{32}, \mathbf{d}_{70}, \mathbf{d}_{90}\} \xrightarrow{\text{merge}} \{\mathbf{d}_0, \mathbf{d}_{31}, \mathbf{d}_{70}, \mathbf{d}_{90}\} = S'$$

Combination: We do not only calculate the error of the whole B-Spline, but also for every interval $[s_i, s_{i+1}]$. When combining two different genes into a new one, we use the following algorithm:

- Iterate over all points \mathbf{d}_j and find the intervals $[s_i, s_{i+1}]$ in both genes that belong to that point.
- Calculate the average error of both intervals.
- Add s_{i+1} of the interval with a smaller average error to S of the new gene.
- Continue with iterating over all remaining points after the previously added point s_{i+1} .

This way, points are added to the new gene which probably had a positive effect on the error of their origin B-Spline. Since the fittest B-Spline in G is combined with all others in G , $|G| - 1$ new genes are generated. Figure 9 shows an example of recombination.

Note that all presented seeding methods and evolution strategies are deterministic. Our tests showed that randomized methods do not compete well against our deterministic strategies and have no impact on the final result.

3.5 Filtering

All new genes created by mutation and combination are collected in a list. After the approximations are calculated and the fitness of each B-Spline is evaluated, the list is sorted by fitness.

Some strategies generate similar genes by definition and often have similar fitness, too. This is true especially for all B-Splines generated by jiggling. When simply merging all new B-Splines back into the database, it probably gets spammed by similar B-Splines. Future threads may then

pick similar genes for further mutation and recombination, producing even more similar results. Overall this can lock the system in a local minimum quite fast.

In order to counter locking in local minima, we remove similar B-Splines from the list and keep a single representative with the best fitness among all similar B-Splines. As similarity function we experimented with the Manhattan Distance of two different B-Splines S and S' which is fast to calculate and leads to desired results. If two B-Splines have a different number of points in S , they are considered different and the similarity function returns the maximum value.

A threshold has to be chosen which determines if a B-Spline is too similar to a different B-Spline with better fitness and thus has to be removed. Choosing 1 as threshold already eliminates all different B-Splines created by jiggling. However, we noticed that a threshold of $\frac{1}{20}$ to $\frac{1}{10}$ produces the best results. After the new B-Splines have been merged into the data base, another filtering iteration is applied to all B-Splines in the database in order to remove similar B-Splines generated by different threads.

4 Results

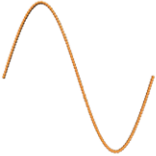
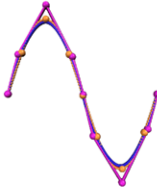
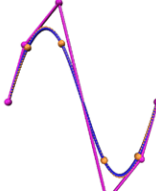
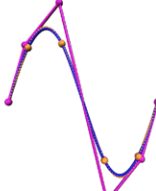
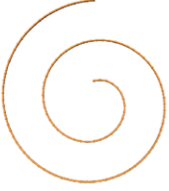
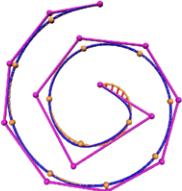
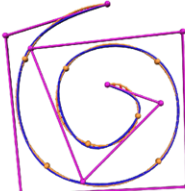
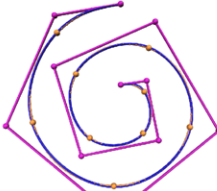


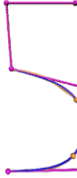
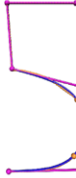

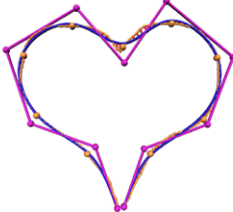

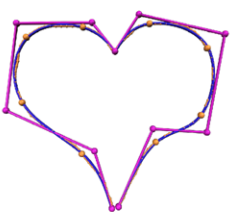


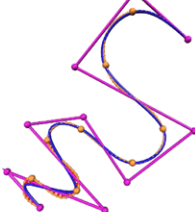
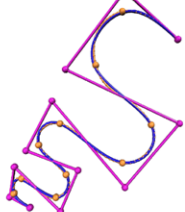
Our tests have been executed on different kinds of CPUs, which include a single core, a dual core and a quad core CPU with hyper threading. While all settings lead to similar results, the fastest results naturally are achieved by the latter one due to the parallelized framework. All results presented in this section are calculated within one second on an i7 920 CPU running at 2.66 GHz. Depending on the complexity of the curve, about 300 to 1500 threads are created and about 40k to 120k approximations are calculated for each result.

Since there is no other publication known to us which uses the same specific B-Spline representation in approximations, we used our uniform and iterative seeding, described in Sect. 3.3, as a substitute for a simple approximation algorithm. These are common B-Spline approximation schemes, presented in various publications with different representations.

For all figures we use the following color scheme: The input curve \mathbf{d}_j is drawn yellow. Knot points s_i are yellow spheres. The B-Spline control polygon \mathbf{p}_i is drawn in purple with spheres for its control points. The B-Spline curve $C(t)$ is drawn in blue. The distances between input and B-Spline curve are illustrated with red lines.

Table 2 illustrates example results of our genetic algorithm. The first column shows the following input curves: A densely sampled sine curve, a densely sampled spiral, a five-shaped hand drawn curve, a heart-shaped hand drawn curve and a serpentine-shaped sampled B-Spline curve. The second column presents the results of the uniform and iterative seeding method. A base of 10 is used in order to get

Table 2 Results for comparison. Columns from left to right: input curve, seeding with a base of 10, our genetic algorithm with a base of 2, our genetic algorithm with a base of 10. Note the great reduction in the number of control points with a base of 2 and the great reduction in the error with a base of 10

input	seeding, $b = 10$	all strategies, $b = 2$	all strategies, $b = 10$
 error ϵ number of control points i fitness function f	 0.0556 9 10.23	 0.0137 6 6.06	 0.0137 6 6.19
 error ϵ number of control points i fitness function f	 0.1176 14 18.35	 0.1322 9 9.86	 0.0154 11 11.4
 error ϵ number of control points i fitness function f	 0.0817 10 12.07	 0.0525 6 6.22	 0.0525 6 6.77
 error ϵ number of control points i fitness function f	 0.1407 11 15.21	 0.1847 9 10.23	 0.0214 11 11.56
 error ϵ number of control points i fitness function f	 0.1259 14 18.71	 0.1544 10 11.13	 0.0276 12 12.79

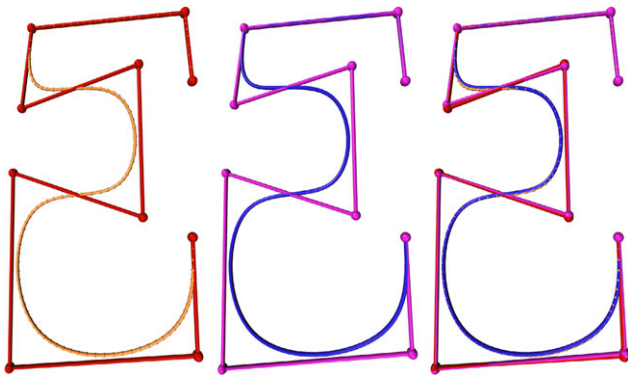


Fig. 10 (left) Sampled B-Spline curve (yellow) with its control polygon (red); (middle) resulting B-Spline (blue curve and purple control polygon) of our algorithm applied to the sampled curve; (right) overlay of both curves and control polygons, demonstrating a small difference

accurate results with a small error. The third column shows results with all evolution strategies enabled and a base of 2. This relatively low base leads to B-Splines with focus on few control points. The fourth column presents results with a base of 10, featuring a small error. Note that with a base of 10 our algorithm provides very accurate results with a much smaller error, while using fewer or equally many control points, compared to B-Splines generated by an usual uniform or iterative method.

As shown in Fig. 10, we also used sampled B-Spline curves as input in order to inspect if our algorithm finds the original B-Spline without any information but its sample points. While our algorithm finds the original knot points of the B-Spline in most cases, the positions of the approximated control points vary slightly. This problem originates from the parameterization in the approximation pipeline. Recall that it is only an estimation of the original parameterization.

An interesting question on genetic algorithms is whether they converge to the same result when repeatedly applied to the same problem. Generally, this is not possible to guarantee and is often not the case. However, our algorithm produced the same B-Splines over and over again. Results naturally start varying if the computation time is reduced, especially on complex input curves. If the filter is loosened or removed, the system may lock in a random local minima. Figure 11 illustrates two slightly different results for the same spiral input curve. In this case, just a few more iterations with the jiggling strategy are needed until the genetic approximation converges to the same B-Spline.

5 Conclusion and future work

In this work, we presented a genetic algorithm which fits a uniform cubic B-Spline to a given densely sampled input curve. The resulting B-Spline has very few control points.

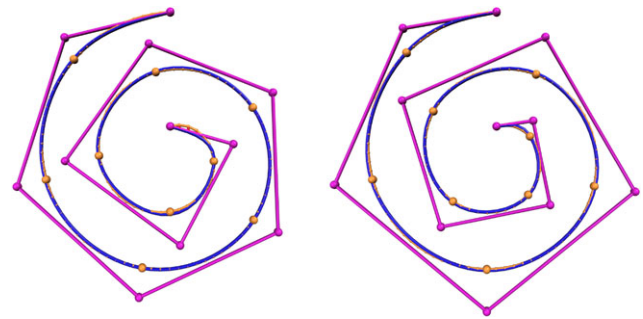


Fig. 11 Two different B-Splines on two runs: (left) fitness $f = 11.732$; (right) fitness $f = 11.401$

Parallelization and adapted evolution strategies have been used to generate results in a short time, suitable for interactive applications.

Examine the filter strategy Our current filter strategy immediately exterminates a gene that is close to an existing gene, if its fitness function is worse. This may prevent some genes from evolving and thus not appearing as a solution. Loosening the filter criteria may lead to local locking of the system. Finding a flexible filter which uses more information than the gene and its fitness may improve the system overall. We experimented with Pareto optimal genes. This means that for a given number of control points n , only one gene is kept within the database. However, we came to the conclusion that this restriction is too harsh.

Dynamic selection of evolution strategies Different stages of exploring the solution space require different strategies. For example, at the beginning coarse scanning may be the most important task. Therefore, small variations like jiggling are not appropriate. A challenge on this topic is creating algorithms that examine the database and decide which strategies to use next.

References

1. Renner, A. Markus G., Vancza, J.: Genetic algorithms in free form curve design. In: *Mathematical Methods for Curves and Surfaces*, pp. 343–354 (1995)
2. Renner, A. Markus G., Vancza, J.: Spline interpolation with genetic algorithms. In: *Proceedings of the 1997 International Conference on Shape Modelling and Applications*, pp. 47–54 (1997)
3. de Boor, C.: *A Practical Guide to Splines (Applied Mathematical Sciences)* (1978)
4. Catmull, E., Clark, J.: Recursively generated b-spline surfaces on arbitrary topological meshes. *Comput. Aided Des.* **10**, 350–355 (1978)
5. Cohen, E., Riesenfeld, R.F., Elber, G.: *Geometric Modeling with Splines: An Introduction* (2001)
6. Farin, G.: *Curves and Surfaces for CAGD: A Practical Guide* (1993)

7. Goldenthal, R., Bercovier, M.: Spline curve approximation and design by optimal control over the knots using genetic algorithms. In: International Congress on Evolutionary Methods for Design, EUROGEN 2003 (2003)
8. Golub, G.H., van Loan, C.F.: Matrix Computations, 2nd edn. The John Hopkins University Press, Baltimore (1989)
9. Havemann, S.: Generative mesh modeling. PhD thesis, Braunschweig Technical University, Germany (2005)
10. Havemann, S., Fellner, D.: Progressive combined b-reps—multi-resolution meshes for interactive real-time shape design. *J. WSCG* **16**(1–3), 121–135 (2008)
11. Hoschek, J.: Intrinsic parametrization for approximation. *Comput. Aided Geom. Des.* **5**, 27–31 (1988)
12. Juhasz, I., Hoffmann, M.: The effect of knot modifications on the shape of b-spline curves. *J. Geom. Graph.* **5**, 111–119 (2001)
13. Laurent-Gengoux, P., Mekhilef, M.: Optimization of a nurbs representation. *Comput. Aided Des.* **25**(11), 699–710 (1993)
14. Renner, G., Ekart, A.: Genetic algorithms in computer aided design. *Comput. Aided Des.* **35**(8), 709–726 (2003). *Genetic Algorithms*
15. Sapidis, N., Farin, G.: Automatic fairing algorithm for b-spline curves. *Comput. Aided Des.* **22**, 121–129 (1990)
16. Shene, D.C.K.: CS3621 Introduction to computing with geometry notes. Department of Computer Science, Michigan Technological University, <http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/>. Last visited 10.02.2011 (2008)
17. Speer, T., Kuppe, M., Hoschek, J.: Global reparametrization for curve approximation. *Comput. Aided Geom. Des.* **15**(9), 869–877 (1998)



Matthias Bein is a PhD Student at the Interactive Graphics Systems Group at the TU Darmstadt. His research activities cover algorithms and software architectures for interactive modeling, sketching techniques, modeling with subdivision surfaces, generative and reconstructive modeling and shape analysis. He received his diploma degree at the TU Darmstadt in 2008 for his thesis “Sketching techniques on Combined B-Reps in GML”. After one year of research in the field of reconstructive modeling at the

Fraunhofer Institute of Computer Graphics (IGD), he started his PhD thesis at the Interactive Graphics Systems Group.



Dieter W. Fellner is a professor at the Interactive Graphics Systems Group at the TU Darmstadt and leads the Fraunhofer IGD in Darmstadt, Germany. He is also a professor at the TU Graz, Austria, where he has established the Institute of Computer Graphics and Knowledge Visualization. The research activities there cover algorithms and software architectures to integrate modeling and rendering, efficient rendering and visualization algorithms, generative and reconstructive modeling, virtual and augmented reality, and digital libraries. His research projects comprise a broad spectrum of areas from formal languages, telematics services, and user interface design, to software engineering, computer graphics, and digital libraries.



André Stork received his doctoral degree from the Darmstadt University of Technology in 2000. Since 2002 he is the head of the Department for Industrial Applications within the Fraunhofer Institute for Computer Graphics in Darmstadt, Germany, where he has been working since 1994. Dr. Stork was involved in or coordinated more than 10 European projects since 2002 and he has authored and co-authored more than 100 papers. His research interests are: interactive modeling, simulation and visualization.