A Local Approach for Computing Smooth B-spline Surfaces for Arbitrary Quadrilateral Base Meshes

Dennis Mosbach[*], **Katja Schladitz** **Bernd Hamann**
Image Processing Department Department of Computer Science
Fraunhofer ITWM University of California
67663 Kaiserslautern, Germany Davis, CA 95616, U.S.A.
Email: dennis.mosbach@itwm.fraunhofer.de Email: hamann@cs.ucdavis.edu
katja.schladitz@itwm.fraunhofer.de

Hans Hagen
Computer Graphics and HCI Group
University of Kaiserslautern
67663 Kaiserslautern, Germany
Email: hagen@cs.uni-kl.de

ABSTRACT

We present a method for approximating surface data of arbitrary topology by a model of smoothly connected B-spline surfaces. Most of the existing solutions for this problem use constructions with limited degrees of freedom or they address smoothness between surfaces in a post-processing step, often leading to undesirable surface behavior in proximity of the boundaries. Our contribution is the design of a local method for the approximation process. We compute a smooth B-spline surface approximation without imposing restrictions on the topology of a quadrilateral base mesh defining the individual B-spline surfaces, the used B-spline knot vectors, or the number of B-spline control points. Exact tangent plane continuity can generally not be achieved for a set of B-spline surfaces for an arbitrary underlying quadrilateral base mesh. Our method generates a set of B-spline surfaces that lead to a nearly tangent plane continuous surface approximation and is watertight, i.e., continuous. The presented examples demonstrate that we can generate B-spline approximations with differences of normal vectors along shared boundary curves of less than one degree.

[*]Corresponding author

Our approach can also be adapted to locally utilize other approximation methods leading to higher orders of continuity.

1 INTRODUCTION

Approximating large quantities of data points by a relatively small number of parametric surfaces is an approach commonly used to reduce data size, re-sample objects or create a model for simulations, analysis, or reverse engineering [1]. A widely used type of surface for this purpose is the tensor product B-spline surface [2]. B-splines are common in various fields, e.g., mechanical and aerospace engineering, computer aided design, and computer graphics. A single B-spline surface is usually not sufficient to represent a complex object. Therefore, it is necessary to design a model consisting of multiple B-spline surfaces. While it is simple to construct a watertight (C^0-continuous) B-spline surface approximation for a set of given data points, achieving higher orders of continuity is still a challenging task. To obtain smooth and visually pleasing models, it is desirable to generate B-spline approximations of at least G^1-continuity, i.e., approximations that have continuous tangent planes everywhere.

This problem has been studied quite intensively, and a wealth of literature is available on the topic. Most existing solutions produce very good results, but introduce constraints, e.g., to the order of the B-splines or the type of knot vector [3], which defines the parameterization of the surface. When the approximation problem must be tackled in a general setting, several established methods have problems, e.g., in case of large or incomplete data. A closely related task is the conversion of a purely quadrilateral base mesh into B-splines. Several local methods exist for constructing G^1-continuously connected B-spline surfaces. Control points are defined as linear combinations of the surrounding quad-vertices [4–6]. When using these vertices as variables in a least-squares approximation approach, this scheme can also be used for data approximation. However, to construct the linear system of a least-squares approach, all control points of the B-spline surfaces must be present in the equation, i.e., locality of the original construction can no longer be utilized.

We propose a local algorithmic framework for the computation of smooth B-spline models to approximate discrete data. Our approach is suitable for B-splines of any order and with arbitrary knot vectors. This allows us to construct a model with low complexity surfaces defined by a small number of control points. The computation of a single surface only requires the data in a small neighborhood to be available. Due to the locality of the approach, it is easy to handle large amounts of data and to parallelly compute individual parts of the result. Most of the existing methods can be localized as well with our method. We demonstrate

its viability by using a computationally more expensive approach for approximate G^1-continuity. Additional to that, we offer suggestions on visualizations and error metrics to evaluate and compare the quality of a B-spline approximation.

The entire process of computing a smooth B-spline model for complex structures consists of several steps. For the sake of completeness, we describe the whole pipeline, although this work focuses on the B-spline approximation step. Data can come from many sources and the first step is to obtain a triangulated surface mesh representing the object of interest. This mesh then gets subdivided into a collection of quadrilateral cells which are parameterized over a rectangular region. The cells of this so called quad-mesh and their parameterization are the starting point for the actual B-spline approximation.

The main idea of our method is to construct smooth approximations of subsets of the data. We compute initial control points where continuity constraints cause dependencies between neighboring surfaces. That way we do not need to solve a global system. First, we construct optimal B-splines around each corner in the quad-mesh and keep the control points necessary for smooth transitions around that corner. With those control points fixed, we then do a spline approximation around each boundary of the quad-mesh, storing only the control points relevant for smooth transitions along that edge. These steps are equivalent to defining a curve network with tangent information. All that is left then is to approximate the interior of each cell in an optimal way. The main advantage of our method is that it only requires data in local neighborhoods, which leads to several small systems of equations that need to be solved instead of one large global system.

We apply this method to represent isosurfaces of binary volume images in the form of B-spline surfaces. To do so, our first step is to apply an isosurface extraction method like Marching Cubes to these images to obtain our initial triangulated mesh. To compensate for the discrete nature of the data, we also apply a Laplacian smoothing to the resulting triangulation.

In Section 1.1 we briefly review relevant existing approaches. Section 2 presents the pipeline and describes the key ideas of our method as well as quality measurements for the resulting models. The computed results are presented in Section 3. In Section 4 we then discuss the advantages as well as the drawbacks of our method. Finally, we provide our conclusions in Section 5.

1.1 Related Work

The construction of smooth B-spline models to approximate or interpolate discrete data is a topic that has been well researched over the past decades. While B-splines are fairly easy to understand and use,

computing a G^1-continuous model of B-spline surfaces is not a trivial task.

Peters and Fan [3] discuss the complexity of constructing smooth piecewise polynomial surfaces and derive essential conditions that need to be met in order to construct a G^1-connected model of B-splines. A central point is that a G^1-continuous B-spline model of arbitrary topology with bi-cubic B-spline surfaces requires at least two internal double knots per edge. Therefore, for a knot vector with only single interior knots it is not guaranteed that there always exists an exact G^1-continuous B-spline model.

However, a variety of methods exist to solve specialized instances. Some accept certain restrictions on the data or B-splines, others aim only for approximate G^1-continuity.

Eck and Hoppe [7] provide a good overview of the overall pipeline to approximate point clouds with B-splines. They first obtain an initial surface mesh which gets partitioned into a coarse quadrilateral mesh. Then, based on an approach by Peters [8] for each cell of the quad-mesh, they calculate the control points of 4×4 G^1-continuous bi-cubic and bi-quadratic Bézier patches, which can together be expressed as a single bi-cubic B-spline surface per cell with multiple interior knots.

Milroy et al. [9] present a stitching approach by defining G^1-errors as difference between normal vectors on a fixed number of points along boundary curves and then minimizing this term. This approach yields good results but is sensitive to the size of the model due to the complexity of the nonlinear constraints.

Lai and Ueng [10] show that even G^2-continuity can be achieved fairly easily when simplifying the continuity constraints to colinearity of derivatives along surface boundaries. Since this leads to singularities when multiple surfaces meet in one point, the method is best suited for constructing pairs or stripes of surfaces.

According to Shi et al. [11] a construction of a G^1-continuous B-spline model with single interior knots requires at least bi-quintic splines. For those, they provide explicit formulas for the computation of the control points. To deal with irregular corner points, they project all outgoing tangents onto the plane defined by an averaging normal vector.

Lin et al. [12] approximate data with bi-quintic Bézier surfaces. They compute a global curve network and generate normal information along those curves. Then, they compute each surface individually such that it interpolates the boundary curves and the normal vectors along those curves to obtain G^1-continuity.

Further work by Lin et al. [13] extends this approach to B-spline surfaces. Their algorithm first defines a network of boundary curves. Subsequently, each surface is established as a bi-cubically blended Coons surface interpolating positions and normal vectors along boundaries. Final B-spline surfaces are created via a warping step that adjusts inner control points to optimally approximate the given data while still preserving

G^1-continuous transitions. The authors discuss improvements of the data processing pipeline to increase overall stability and quality of the quad-meshes.

Yoo [14] presents an approach that utilizes a global implicit surface to generate a quad-mesh in addition to sample points that are then interpolated by one B-spline per quad. Smoothness between individual B-splines is obtained by also interpolating the normal vectors of the implicit surface along the surface boundaries.

Zhao et al. [15] introduce a numerical approach for point cloud approximation. First, they approximate the data using a set of B-spline surfaces with no continuity constraints, even allowing different knot vectors. Then, the control points are adjusted by numerical methods to also obtain approximately C^0 and G^1-continuous transitions.

Closely related to the task of approximating data is the interpolation problem, where the vertices of a given (quad-)mesh are to be interpolated by a number of smooth surface patches. These approaches need to deal with the same continuity constraints and can be modified to also allow data approximation.

Hahmann et al. [4] describe a localized method to interpolate quad-meshes by using 2×2 bi-cubic Bézier patches per quad. Their algorithm consists of three steps, calculating control points around corners and edges first to ensure G^1-continuity there, before defining the interior control points. However, this method cannot be applied to arbitrary meshes which is then addressed by a more general approach by Bonneau and Hahmann [5] using a similar idea of a localized construction to obtain 2×2 bi-quartic Bézier patches per quad.

Yoshihara et al. [16] show how to make use of an interpolation method for data approximation. They approximate the data with a Catmull-Clark subdivision surface. The control structure is then projected onto its limit surface to obtain a dense quad-mesh. Vertices of that mesh are then interpolated using an approach by Fan and Peters [17] which computes a model of 3×3 bi-cubic Bézier patches per quad.

Fan and Peters also present an improved version of that algorithm [6] where they construct either a single bi-cubic Bézier patch or a B-spline surface with at most two inner double knots per quad, depending on the valence of the surrounding vertices. This construction even provides C^2-continuity along regular edges, i.e., edges that connect vertices with valence 4.

A more formal approach to the problem of constructing smooth splines has been proposed by Mourrain et al. [18]. They provide a detailed analysis and in-depth theoretical background on the dimensionality and bases of the space of smoothly connected splines. While they look at general splines defined over arbitrary topologies, Blidia et al. [19] provide a similarly thorough investigation of spline spaces on quad meshes with

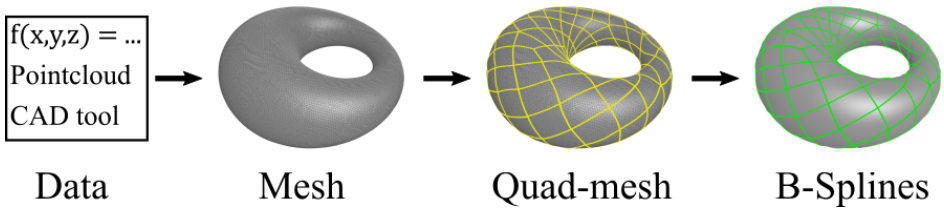


Fig. 1: B-spline approximation pipeline. Original points are triangulated to represent a surface's geometry and topology. The triangle mesh is partitioned into a quad-mesh. Finally, we compute a B-spline surface for each quad-mesh cell.

4-split elements.

Most of the above mentioned methods can be used to generate very good, smooth B-spline surfaces with certain restrictions, e.g., the spline order is often fixed and by using Bézier patches they already define the knot vector and number of control points. Our approach aims to provide a more general solution for data approximation and we impose no restrictions on the topology of the quad-mesh, the order of the B-splines, or the used knot vectors.

2 METHOD

In the following, we assume that our input is given in the form of a dense surface mesh. We only consider the vertices of that mesh for the actual B-spline approximation, but the topological information given by the mesh structure is relevant for the first steps in the processing pipeline. While it is not a requirement for our method, we use triangular meshes here.

2.1 Overall Pipeline

The goal is to create a set of B-spline surfaces that represent our input mesh. This process consists of several steps and there are several ways to perform each of them. Here, our initial surface is defined by a set of triangles. First, we compute a partition of this mesh into quadrilateral cells, called a *quad-mesh*, where each data point (i.e., vertices) of the initial mesh gets assigned to a cell in the quad-mesh as well as to an initial parameter value $(u, v) \in [0, 1]^2$. Then, we can apply our method to compute a model of smoothly connected B-spline surfaces. Figure 1 illustrates that pipeline.

2.2 Quad-Mesh Generation

Regular tensor product B-spline surfaces are defined over a rectangular parameter domain and a single surface can only represent limited topologies. In order to represent complex objects with B-spline surfaces,

it is common practice to subdivide these objects into rectangular regions [7, 9, 11, 12, 14–16]. This can be done manually by using software to draw boundary curves directly onto the surface, but it becomes impractical for big or too complex objects of interest.

To provide a pipeline in a way that requires as little user interaction as possible, we use a simple approach based on [20] where a surface Voronoi tessellation gets computed on the mesh and then refined such that each cell has the topology of a disk. These arbitrary-sided Voronoi cells then get rearranged into four-sided quadrilateral cells, called quad-cells or quads. We then assign parameter values (u_i, v_i) to each data point p_i within a cell via harmonic maps. Such and other advanced implementations of mesh parameterization methods are included in the freely available package CGAL [21].

Note that the shape of the quad-mesh clearly has an impact on the quality of the resulting B-splines as we briefly demonstrate in Section 4.2. Depending on the application, an experienced human user could probably design a better quad-mesh than most automatic algorithms. Finding a good quadrilateral partitioning for arbitrary shapes is still a challenging task. However, recently developed algorithms, e.g., Quadriflow [22], offer a lot of versatility in solving it.

2.3 B-spline Approximation

We consider B-splines of general order k and, for simplicity, assume that the parameter domain of each surface is the unit square $[0,1]^2$. To avoid complicated continuity constraints for surface transitions later, we restrict ourselves to quadratic grids of $n_c \times n_c$ control points and require that the knot vectors $\tau = (0, ..., 0, \tau_1, ..., \tau_{n_c-k-1}, 1, ..., 1) \in [0,1]^{n_c+k+1}$ are equal for all surfaces in all directions. While the inner knots can be chosen arbitrarily, we require $k+1$ multiple knots at the beginning and end of τ to get the endpoint interpolation property. We define a B-spline tensor product surface by the following mapping:

$$S : [0,1]^2 \to \mathbb{R}^3$$
$$S(u,v) = \sum_{i=1}^{n_c} \sum_{j=1}^{n_c} N_i(u) N_j(v) b_{i,j}$$

with control points $b_{i,j} \in \mathbb{R}^3$ and $N_i(u) := N_{i,k,\tau}(u)$ being the i-th B-spline basis functions of order k assigned to knot vector τ. By denoting the matrix of control points as a vector $b = (b_{1,1}, ..., b_{n_c,n_c})^T$ we can later express many terms as simple matrix-vector products. More information about B-splines in general can be found in literature [2, 23].

In order to construct a B-spline surface that closely approximates a set of given data points p_i, $i = 1, ...,$ it is common practice to apply a least squares method with respect to Euclidean distance [2]. The goal is to minimize the sum of the squared errors

$$E_{\mathsf{LS}} = \sum_i d_i^2 \tag{1}$$

with the error d_i of point p_i being the shortest distance between that point and the B-spline surface S

$$d_i = \min_{u,v \in [0,1]} \|S(u,v) - p_i\|. \tag{2}$$

Calculating the error E_{LS} is a nonlinear problem. In practice, it is simpler to approximate it iteratively [7, 24]. First, we parameterize our data points, i.e., assign each point p_i some parameter values u_i, v_i. We can define the error of the approximation by

$$d_i \approx \|S(u_i, v_i) - p_i\|. \tag{3}$$

Using this approximation, the only unknowns in (1) are the B-spline control points. The system is quadratic with respect to all the unknowns and an optimal solution can be easily computed.

Once a surface S has been computed, we can find better parameter values (u, v) for each data point by projecting them perpendicularly onto the surface [24], improving the approximation of (2). We can then alternate between surface construction and parameter optimization until the resulting changes are sufficiently small.

To obtain smooth surfaces, we add an energy term to our minimization problem. A commonly used approach for this purpose is based on the *thin-plate-energy* [7, 25], describing the bending energy of a thin

plate as

$$E_{\mathsf{TP}} = \int_0^1 \int_0^1 \left\| \frac{\partial^2}{\partial u^2} S(u,v) \right\|^2 + 2 \left\| \frac{\partial^2}{\partial u \partial v} S(u,v) \right\|^2 + \left\| \frac{\partial^2}{\partial v^2} S(u,v) \right\|^2 \mathrm{d}u\mathrm{d}v.$$

For B-splines, it can be re-written as

$$E_{\mathsf{TP}} = b_x^T M b_x + b_y^T M b_y + b_z^T M b_z$$

with b_x, b_y, b_z being the individual x, y, z components of the B-spline control point vector and M being a matrix that only depends on the number of control points, knot vector, and order of the B-spline. If we intend to construct several surfaces where these characteristics are equal, we will have to compute M only once.

A smooth B-spline surface can be computed by minimizing $\lambda_{\mathsf{LS}} E_{\mathsf{LS}} + \lambda_{\mathsf{TP}} E_{\mathsf{TP}}$ with $\lambda_{\mathsf{LS}} + \lambda_{\mathsf{TP}} = 1$.

In practice, a single surface is usually not enough to represent complex objects. In order to use several surfaces we need to introduce constraints to ensure continuity along the boundaries.

A watertight (i.e., C^0-continuous) model can be easily obtained by ensuring that the control points of two B-spline surfaces along a shared boundary curve are equal. If we store the control points of all surfaces in one vector, we can simply write the constraints for the entire surface model as $Gb = 0$, with G being a sparse matrix where each row contains a constraint of the form $b_i - b_j = 0$.

To generate a smooth model of B-spline surfaces, we also aim for G^1-continuous transitions, i.e., tangent plane continuity. Formulating these constraints analytically along the boundaries between all surfaces leads to a large system of nonlinear equations which is not trivial to solve. Since our proposed method reduces the size of the problem by only looking at local neighborhoods, we choose to use pointwise nonlinear G^1-constraints here. Note, that our method can also be applied with simplified and faster constraints used in the approaches discussed in Section 1.1.

Without loss of generality, assume that two surfaces S_a and S_b meet at $S_a(1,t) = S_b(0,t), t \in [0,1]$.

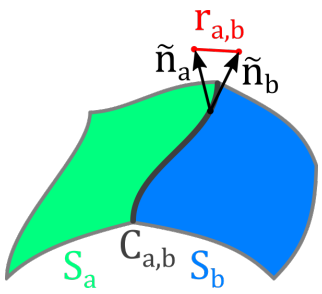


Fig. 2: The G^1-error $r_{a,b}$ at a point on the boundary of two B-spline surfaces is defined as the length of the difference vector between the unit-normal vectors $\tilde{n}_a$ and $\tilde{n}_b$.

Similar to [9], we define the G^1-error along the shared boundary curve as

$$r_{a,b}(t) = \left\| \frac{n_a(1,t)}{\|n_a(1,t)\|} - \frac{n_b(0,t)}{\|n_b(0,t)\|} \right\|, \tag{4}$$

where n_a and n_b are the normal vectors of the two surfaces. Figure 2 provides an illustration of the G^1-error at a point on the shared boundary between two surfaces.

Since the B-spline basis functions and their derivatives are piecewise polynomial functions, we can expect the G^1-error to be low between two points with $r_{a,b}(t_1) = r_{a,b}(t_2) = 0$ as long as $|t_1 - t_2|$ is sufficiently small. We are not aiming for a perfect G^1-continuous construction here, so we only set G^1-constraints for discrete points along the surface boundaries, i.e., for each pair of surfaces that need to be joined G^1-continuously, we add n_{G^1} constraints to our system of equations in the form of $r_{a,b}(t_i) = 0$. By choosing $t_i = i/(n_{G^1} - 1)$ independently of the knot vector, we ensure that these constraints are enforced evenly along surface boundaries. However, doing this for the global system with many B-spline surfaces at once would be computationally impractical. We address this issue in Subsection 2.4.

Let Γ be the set of all index tuples (a, b) of surfaces that are supposed to be joined with G^1-continuity. The minimization problem to be solved can be described as follows:

$$\begin{aligned} &\text{Minimize } \lambda_{\text{LS}} E_{\text{LS}} + \lambda_{\text{TP}} E_{\text{TP}} && \\ &\text{s.t. } Gb = 0 && \\ &\quad r_{a,b}(t_i) = 0 && \forall (a,b) \in \Gamma, i = 1, ..., n_{G^1} \end{aligned} \tag{5}$$

For implementation purposes, the G^1-constraints can be reformulated as $(r_{a,b}(t_i))^2 = 0$, which simply removes the outer square root introduced by taking the norm in equation (4).

As stated in [3], a perfect G^1-continuous construction may not always be possible in a general setting. Also, nonlinear constraints can be hard to handle for some numerical solvers. To compensate, it is also possible to add a weight to the G^1-constraints and move them to the objective function, leading to the following formulation:

$$\begin{aligned} &\text{Minimize } \lambda_{\mathsf{LS}} E_{\mathsf{LS}} + \lambda_{\mathsf{TP}} E_{\mathsf{TP}} + \lambda_{G^1} \sum_{(a,b)\in\Gamma} \sum_{i=1}^{n_{G^1}} r_{a,b}(t_i) \\ &\text{s.t. } Gb = 0 \end{aligned} \tag{6}$$

To handle non-linearity introduced by the G^1-constraints, we use an implementation of sequential least-squares quadratic programming (SLSQP) [26], freely available in the NLopt package [27]. This algorithm deals with nonlinear constraints. However, it does not guarantee convergence to an optimal solution. To obtain a "good-enough" solution, we first solve the linear system obtained when leaving out G^1-constraints. Using this C^0-continuous surface model as initial solution for the nonlinear optimization allows the algorithm to primarily focus on adjusting the tangents along surface boundaries. If a local minimum is reached, the result is thus still viable.

2.4 Local Multi-step Algorithm

The first derivatives of B-spline surfaces along their boundaries only depend on two rows of control points, i.e., $b_{i,j}$, with one index being in $\{1, 2, n_c - 1, n_c\}$. Hence, inner control points can be chosen freely without affecting G^1-continuity. We call the control points that have an influence on the derivatives along a surface boundary *G^1-relevant control points* for that boundary. At corner points of a surface where two boundaries meet, both tangents in u and v direction are defined by only three control points, i.e., the control point in that corner as well as one control point in each direction. The first inner control points in each corner, the so-called twist points, play a special role. They do not have an impact on the normal vector in the corner itself but along both boundaries in a neighborhood around the corner; therefore, they must be chosen to ensure compatibility with both neighboring surfaces, see Figure 3. As we show in Section 3, using these points alone already produces good results. If the goal is exact G^1-continuity, it will also be necessary to consider the so-called curvature points, i.e., the second control point from each corner on the

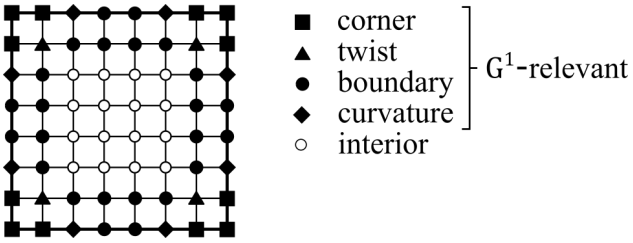


Fig. 3: Control points and G^1-continuity. The figure shows the control points relevant for ensuring G^1-continuity of a B-spline control mesh with 8×8 control points.

boundary, as they are also a part the G^1-compatibility constraints discussed in [3].

The main idea of our local algorithm is similar to other approaches [4–6, 11] that construct a global result by performing various computations in subsequent phases. Instead of computing only individual control points in a phase, our approach computes entire surfaces that approximate small subsets of the data. Relevant control points of these surfaces are used in the final result, while the rest is discarded and re-computed in the next step. The individual phases are:

1. *Corner Phase*

   For each corner of the quad-mesh, we compute C^0-continuously connected B-spline approximations of the incident cells while ensuring G^1-continuity around that corner. We store the G^1-relevant control points around the corner for the final result, i.e., assume that the control points of the surrounding B-spline surfaces are aligned such that $b_{0,0}$ is associated with the corner currently being considered. We then keep the points $b_{0,0}$, $b_{0,1}$, $b_{1,0}$, and $b_{1,1}$ of all these surfaces.
2. *Boundary Phase*

   For each boundary between any two quad-cells, we construct a G^1-continuous B-spline approximation of those cells. The G^1-relevant control points around the end points of the boundary are set as the result of the last phase (corner phase). All other control points can be chosen freely again. We keep the G^1-relevant control points along the shared boundary for the final result.
3. *Interior Phase*

   For each cell of the quad-mesh, we perform a B-spline approximation step, considering all G^1-relevant control points already known from the previous two phases. We use the interior control points to finalize the global result.

The specific spline approximation algorithms used in the individual phases can be chosen freely. We use the objective function and constraints from Section 2.3 to demonstrate the viability of this approach.

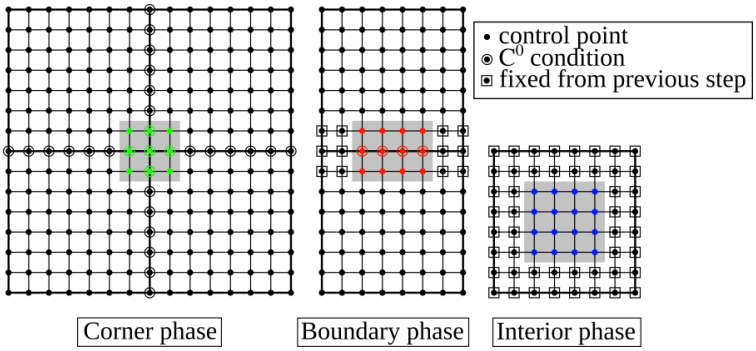


Fig. 4: Phases of control point computation. The individual images high-light the control points used in subsequent phases.

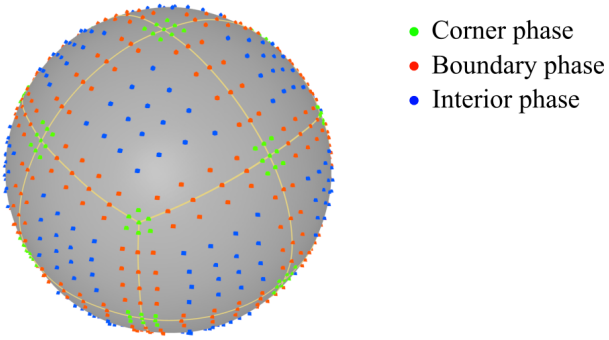


Fig. 5: Final B-spline control meshes. The figure shows the contributions of control points produced by the three phases. Control point colors indicate the phases that generated them.

Section 1.1 provides a discussion of other viable algorithms. These methods can also be used in our local framework. To assess the quality of the resulting surface approximation, where only approximate G^1-continuity is achieved, we describe evaluation criteria in Section 2.5. Our method does not restrict knot vectors, spline order, or number of control points. However, a minimum of 5×5 control points is required to avoid incompatibilities during the corner phase and have available degrees of freedom in the subsequent phases.

We achieve approximate G^1-continuity around the corners in the first phase by numerically solving (5) or (6), using constraints of the form $r_{a,b}(t_0) = 0$, with $t_0 \in \{0, 1\}$ being the parameter value corresponding to that corner for all boundaries between neighboring surfaces S_a and S_b incident to that corner. Each control point with index i has an impact on the interval $[\tau_i, \tau_{i+k+1})$. Since the twist control points are computed during the corner phase as well, we want to increase smoothness along the affected interval; otherwise, it can lead to incompatibilities with the results of the next phase. We achieve this by evenly distributing n_{G^1}

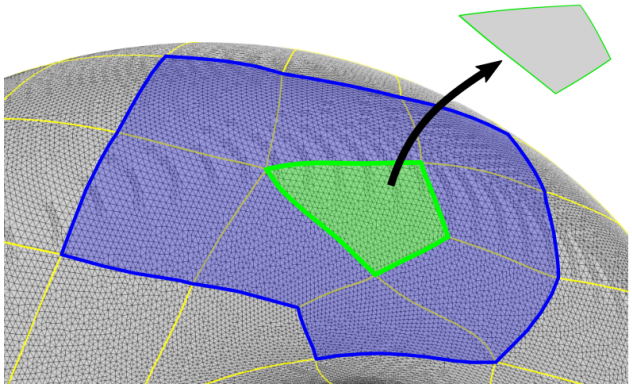


Fig. 6: To construct a B-spline surface that approximates the data of one cell (green), only the data of neighboring cells is needed (blue).

point-wise G^1-constraints along the entire interval influenced by the corner and twist points. In addition to the G^1-constraints at a corner, the surfaces are C^0-continuously connected everywhere else, i.e., the rows of control points along shared boundaries must be identical.

When computing B-spline approximations during the boundary phase, we again solve (5) or (6) with n_{G^1} evenly distributed G^1-constraints along the boundary. Here, we exclude the corner points, as they are not affected by the free control points.

A B-spline surface computed by the interior phase is the final approximation of the corresponding quad-cell. To compute a surface in this phase the results of the corner and boundary phases affecting that cell are needed. Only the data of the neighboring quad-cells needs to be known at this time, see Figure 6. This means that the individual phases do not have to be completed for the entire quad-mesh before starting surface computation in the final phase. This fact makes it possible to apply this approach even when only parts of the data can be loaded into memory.

2.5 Quality Evaluation

Depending on the application it might suffice if the constructed B-spline model passes a visual inspection. However, for a formal evaluation, well-defined quality criteria are desirable.

A good error measure to evaluate a spline approximation is the distance to the original object/surface. We define the *approximation error* of each data point as the distance to the closest point on the B-spline surface, see (2). Due to the iterative scheme described in Section 2.3, we expect to find the closest surface point approximately by evaluating the B-spline at the parameter values of the data point, i.e., by evaluating (3). To obtain the most accurate result for evaluation purposes, we can also compute the closest surface

points by numerically solving the nonlinear equation (2) to compute optimal parameter values (u, v). We measure the distances between the B-splines and the given data points, which are the vertices of the input mesh. When these data points are obtained via a more complex pipeline, pre-processing steps can add additional error which has to be considered separately.

To analyze the approximation error of the whole surface, we can either use statistical measures of the individual errors or support a qualitative analysis by plotting the error values, color-coded onto the B-spline surfaces, see Figure 7a. When the goal is to smooth out irregularities in the original data, we expect the plot to show higher errors in these irregular regions. To understand where data points are above or below the approximating B-splines we compute a signed error value defined by the scalar product between the error vector of a data point and the normal vector at its closest point on the B-spline surface.

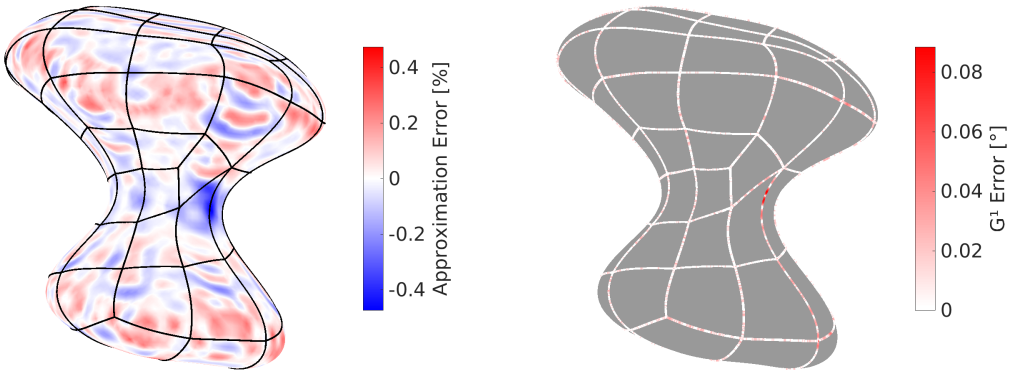
In order to establish a statistical measure for overall approximation quality, we compute the root mean square (RMS) error

$$E_{\mathsf{RMS}} = \sqrt{\frac{1}{n_d} \sum_{i=1}^{n_d} d_i^2},$$

with n_d being the total number of data points. Furthermore, we measure the maximum error. To enable a meaningful comparison between data sets of different sizes, we define the error as a percentage of the "diameter" of the data set, defined as the largest distance between any two data points, see [7, 16].

When an input mesh already exhibits undesirable geometrical behavior, e.g., extensive variation of normal vectors of the triangles resulting from an iso-surfacing method, like the Marching Cubes method, these effects have an impact on statistical quality measures and color plots. Since these effects are usually a by-product of an earlier step in the data processing pipeline, they generally do not reflect the proper geometry of an original object. A good spline approximation scheme eliminates or greatly reduces such undesirable irregularities, but, numerically, this smoothing effect impacts error computation. Ultimately, a user must interpret quantitative error values and compare results with the input.

In addition to the actual approximation error, we consider smoothness in the form of tangent plane continuity. In a surface model composed of B-spline surfaces with non-degenerate control points and knot vectors, the only regions where discontinuities in first derivatives can occur are along the boundary curves between individual surfaces. Since an exact G^1-continuous model cannot always be constructed in a general setting [3], we must define a meaningful measurement to evaluate smoothness along boundary curves.



(a) Approximation error shows surface areas above or below original data.

(b) G^1-error shows how much normal vectors differ on surface boundaries.

Fig. 7: Error visualization. Error values mapped onto the surface support a qualitative evaluation of an approximation.

Based on equation (4), we consider the normal vectors of two adjacent surfaces along their shared boundary curve. We define the G^1-*error* as the angle between these vectors. These values are computed along all surface boundaries. The result is then color-coded and visualized on the surface enabling an interactive visual assessment of the result, see Figure 7b.

To calculate meaningful measures that summarize the G^1-error of the entire model via a single number, we compute the average G^1-error obtained by integrating the G^1-error of the entire surface model and dividing it by the total length of all boundary curves. Let $C_{a,b}(t)$ be the shared boundary curve between B-spline surfaces S_a and S_b. Since $C_{a,b}(t)$ is usually not parameterized by arc length, the parameter interval $[0, 1]$ can get distorted in $\mathbb{R}^3$. Therefore, simply computing a G^1-error in the parameter domain by using the integral $\int_0^1 r_{a,b}(t)dt$ does not generally yield an acceptable result for the error along the curve. Hence, we integrate along the curve in $\mathbb{R}^3$. We formally extend the definition of $r_{a,b}(t)$ to the 3D setting by defining $r_{a,b}(C_{a,b}(t)) := r_{a,b}(t)$. We can reformulate the integral along a curve parameterized over $[0, 1]$ using $\int_{C_{a,b}} g(C_{a,b})ds = \int_0^1 r_{a,b}(C_{a,b}(t))\|\dot{C}_{a,b}(t)\|dt$. To normalize the result, we divide by the length of all boundary curves $L = \sum_{(a,b)\in\Gamma} \int_0^1 \|\dot{C}(t)\|dt$. The average G^1-error of the system can be defined as

$$E_{G^1} = \frac{1}{L} \sum_{(a,b)\in\Gamma} \int_0^1 r_{a,b}(t) \left\|\dot{C}_{a,b}(t)\right\| dt.$$

As an upper error bound, we also compute the maximum G^1-error among all points on the surface model.

Table 1: Approximation and G^1-error values for test data sets of a C^0-continuous approximation method (left), our method (middle), and a G^1-continuous method (right).

	C^0-Method				**Our Method**				G^1-Method	
	Approx. Error [%]		G^1-Error [°]		Approx. Error [%]		G^1-Error [°]		Approx. Error [%]	
Object	RMS	Max	Avg	Max	RMS	Max	Avg	Max	RMS	Max
Sphere	0.004	0.025	0.522	2.305	0.006	0.032	0.006	0.048	0.048	0.359
Cube	0.092	1.330	3.795	83.955	0.154	1.883	0.009	0.112	1.050	6.176
Spring	0.013	0.110	3.132	30.640	0.015	0.127	0.007	0.108	0.219	1.191
Torus	0.006	0.046	1.271	9.482	0.008	0.055	0.005	0.059	0.074	0.418
As. Torus	0.007	0.074	2.786	11.181	0.005	0.034	0.005	0.091	0.060	0.399
Tanglecube	0.007	0.080	3.436	21.076	0.007	0.077	0.011	0.207	0.168	1.306
Stone	0.041	0.737	7.000	84.278	0.066	0.891	0.007	0.099	0.430	2.445
Fibers	0.002	0.018	5.133	31.278	0.001	0.031	0.016	0.292	0.024	0.361

3 RESULTS

We have applied our method to several synthetic and real-world data sets. To evaluate the quality of the resulting B-spline models, the approximation and G^1-errors are measured as described in Section 2.5. The B-spline models are defined by 6×6 control points per surface, and a uniform knot vector with multiple end knots, i.e., $\tau = (0, 0, 0, 0, \frac{1}{3}, \frac{2}{3}, 1, 1, 1, 1)$. Our experiments have shown that the choice of $\lambda_{\mathsf{TP}} = 0.05$ for the smoothness term leads to good results.

We have compared our results to models that are only C^0-continuous. These models are obtained by minimizing (5) without G^1-constraints. Since no deformations are imposed by the G^1-continuous transitions, we have reduced the weight of the smoothness term to $\lambda_{\mathsf{TP}} = 0.01$.

We have also compared our method to the G^1-exact construction by Fan and Peters [6]. Their approach derives the positions of control points as linear combinations of the vertices of a given quad-mesh. Using these vertices as variables, the global system (5) is used to define a minimization problem, without constraints, to find an optimal solution. Due to the restrictions imposed by exact G^1-continuity, we use the required minimal number of 8×8 control points per B-spline surface and define the knot vector as $\tau = (0, 0, 0, 0, \frac{1}{3}, \frac{1}{3}, \frac{2}{3}, \frac{2}{3}, 1, 1, 1, 1)$. Since the resulting surface model is constructed to be G^1-continuous, we measure only approximation error.

Table 1 provides the values of the quality measures of our algorithm and the two reference methods.

We obtained the initial mesh by using the Marching Cubes algorithm to extract isosurfaces from given

binary 3D volume images. A Laplacian smoothing operation was applied to the mesh to compensate for discretization effects and artifacts resulting from the piece-wise linear interpolation underlying the Marching Cubes method. We tested our method for several synthetically generated models with different geometry and topology and two real-world, application-driven data sets. The first real-world data set is a stone particle used in composite materials. It has simple topology, and we are mainly interested in the geometry of its surface. The second real-world data set consists of two steel fibers found in fiber-reinforced concrete. The fibers' surfaces are quite smooth, and we are primarily interested in overall shape and connectivity of objects. Both data sets were given as 3D images, generated via scanning the objects with micro-computed tomography (Fraunhofer ITWM). Figures 8-10 show the initial meshes, our resulting B-spline approximations (including the subdivisions into individual surfaces), and color plots visualizing approximation error.

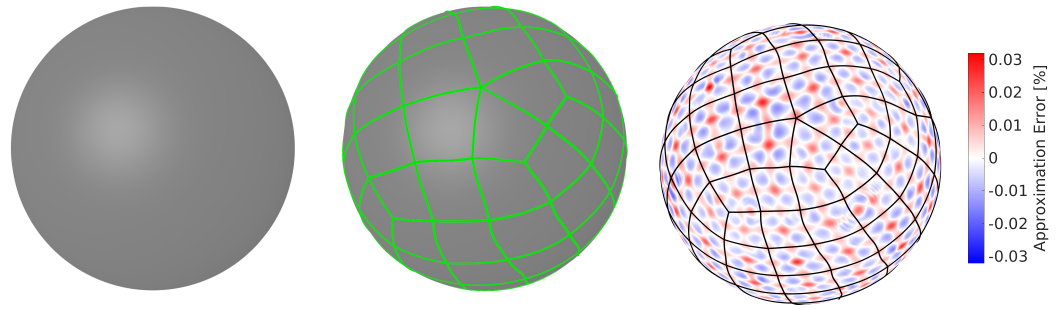
4 DISCUSSION

The results provided in Table 1 show that our method provides a good trade-off between an accurate reconstruction of the data and tangent plane continuity along shared boundaries. Approximation error values compare well to those of the models with only C^0-continuity. Our comparison also shows that even though the G^1-continuous models consist of more control points, satisfying the restrictions needed for exact G^1-continuity severely limits the available degrees of freedom when approximating a dense set of points. As a consequence, the RMS error is about a factor of 10 higher than that of our method.

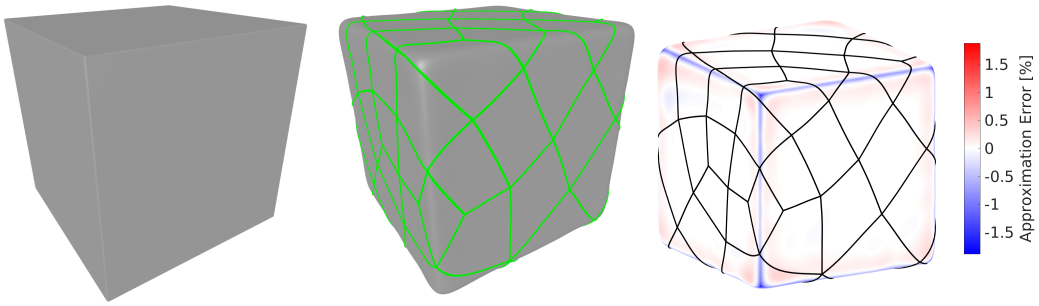
Reconstructing the cube model (Figure 8d) shows how sharp edges become smooth corners in the B-spline model. The existence of such features in the B-spline model violates the definition of G^1-continuity. However, when the boundaries of the quad-mesh align with edges of the object, our approach can be modified to omit the G^1-constraints in these locations and reconstruct these features while still producing a smooth model everywhere else.

4.1 Complexity

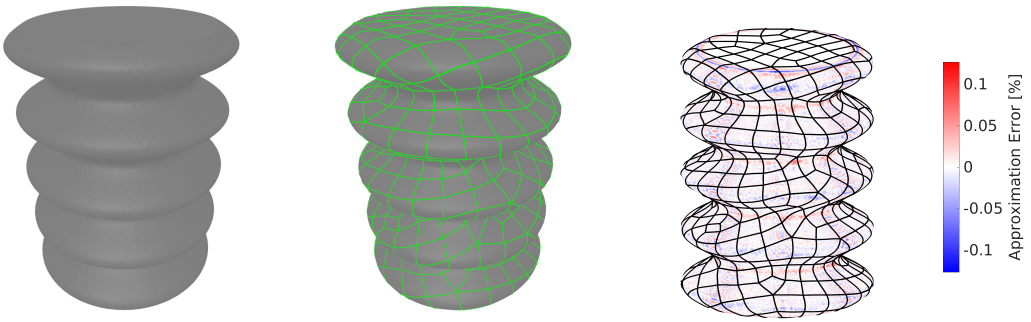
When computing control points in a local neighborhood around corners and boundaries of the quad-mesh, we compute B-spline approximations for all incident quad-cells, even though only a subset of the control points is contributing to the final result. The benefit of this approach becomes evident when considering the computational complexity of the algorithm. Assume that $t(n)$ describes the computational cost needed to perform a G^1-continuous approximation with n B-spline surfaces. Let V, E, F be the number of corners (vertices), boundaries (edges), and cells (faces) of the quad-mesh.



(a) Triangle mesh ($753\,774$ vertices) (b) B-spline model (130 surfaces) (c) Approximation error



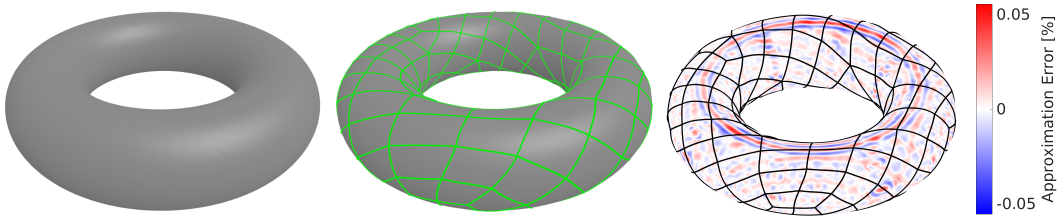
(d) Triangle mesh ($48\,600$ vertices) (e) B-spline model (60 surfaces) (f) Approximation error



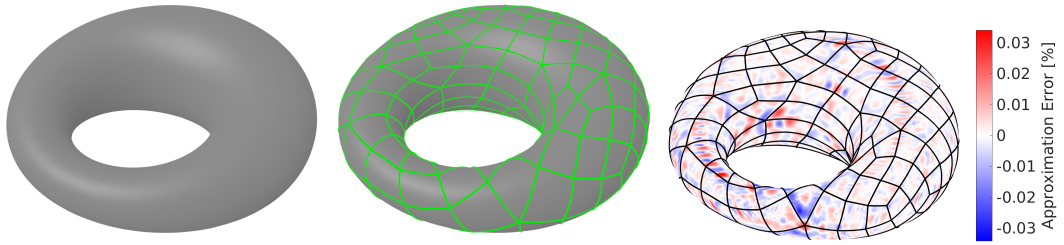
(g) Triangle mesh ($861\,858$ vertices) (h) B-spline model (538 surfaces) (i) Approximation error

Fig. 8: Sphere (top), cube (middle), and spring (bottom). These examples share a simple topological situation. However, their geometrical complexity varies. The sphere has constant curvature while the cube consists of several flat surfaces. The spring consists of regions of strongly varying curvature. Errors are highest close to edges and regions of changing curvature. Our method was designed to construct a smooth surface model and it consequently smoothes sharp edges.

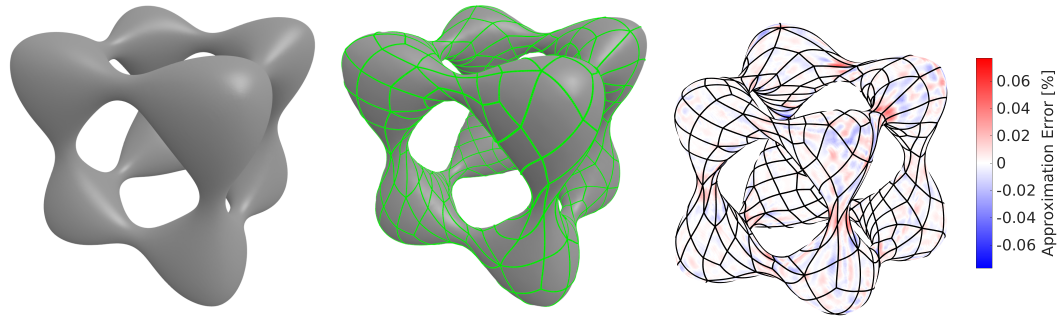
In the quad-meshes we obtained in our experimental tests, no corner had a valence above 8. In theory that number could be reduced even more by applying optimization algorithms to generate a more even distribution of valence values. Let n_v be the highest valence of any corner in the quad-mesh. The largest systems that has to be solved during the B-spline computation consists of at most n_v surfaces, i.e., the



(a) Triangle mesh ($426\,224$ vertices) (b) B-spline model (180 surfaces) (c) Approximation error



(d) Triangle mesh ($151\,954$ vertices) (e) B-spline model (248 surfaces) (f) Approximation error

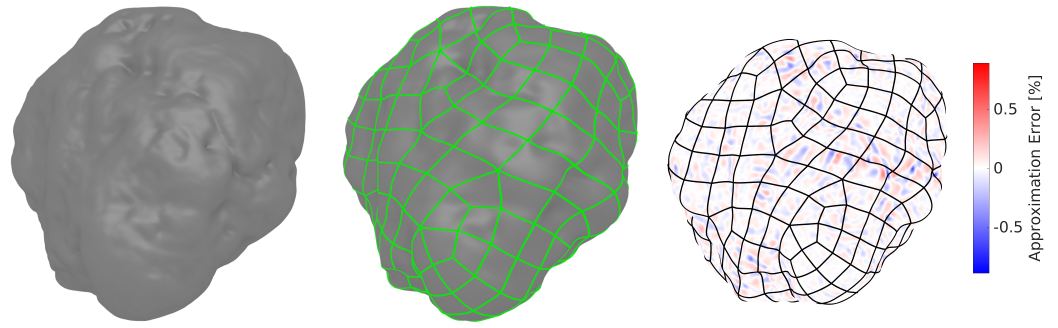


(g) Triangle mesh ($383\,718$ vertices) (h) B-spline model (452 surfaces) (i) Approximation error

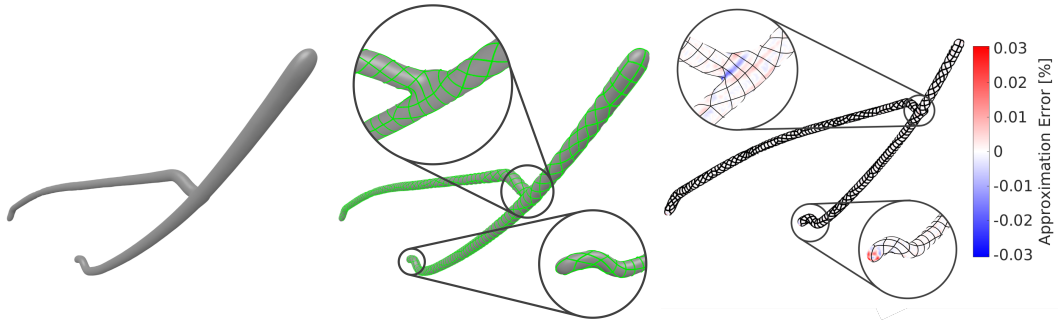
Fig. 9: Torus (top), asymmetric torus (middle), and tanglecube (bottom). Even with increasing topological complexity, our method still produces high-quality results. Errors are highest in regions where structures transition from strong to weak surface bending.

computational cost for the first phase is at most $V \cdot t(n_v)$.

During the boundary phase we consider only pairs of neighboring quad-cells. Assuming we ignore potential speed-up resulting from having already a number of fixed control points from the first phase, the computational cost of the boundary phase is $E \cdot t(2)$.



(a) Triangle mesh ($240\,468$ vertices) (b) B-spline model (234 surfaces) (c) Approximation error



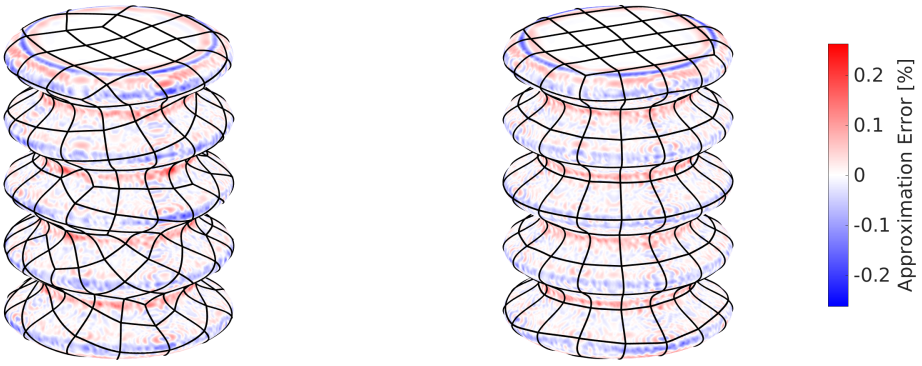
(d) Triangle mesh ($439\,464$ vertices) (e) B-spline model (718 surfaces) (f) Approximation error

Fig. 10: Stone (top) and fibers (bottom). Sharp geometrical features are smoothed out. The overall approximation error is still low, and important shape characteristics are well-preserved.

Finally, in the interior phase, we treat every G^1-relevant vertex as a constant. We must find a solution for only $(n_c - 4)^2$ unknown control points. To provide a more intuitive upper bound of our method, we assume that the computational cost of the final phase is $F \cdot t(1)$.

This leads to a total computational cost of at most $V \cdot t(n_v) + E \cdot t(2) + F \cdot t(1)$. Solving a global system would have a cost of $t(F)$. If it were possible to define a system that scales linearly, i.e., $t(F) = F \cdot t(1)$, it would be computationally cheaper to solve the system globally. Unfortunately, to our knowledge such an approach does not exist yet. Usually, $t(n)$ is at least in $\Omega(n \log n)$ or worse. We note that the same argument also holds for memory requirements. The total amount of memory required to represent the systems of equations for several small systems is by far less than the amount required for a global system.

Besides having a lower runtime and memory complexity, our local approach can also make use of parallel computing. In practice, this means that we could speed up the computation at the cost of higher memory requirements, allowing the system to adapt its performance to the available hardware.



(a) No. of quads: 258
RMS error: 0.037%
Max error: 0.262%
Avg G^1: $0.014°$
Max G^1: $0.171°$

(b) No. of quads: 224
RMS error: 0.033%
Max error: 0.205%
Avg G^1: $0.005°$
Max G^1: $0.079°$

Fig. 11: Impact of quad-mesh. (a) Quad-mesh created automatically; (b) quad-mesh generated manually with cell boundaries in principal curvature directions.

4.2 Impact and Limitations of the Quad-Mesh

We have demonstrated the viability and robustness of our method by using quad-meshes that do not align with geometric features. However, the error visualizations indicate that the quality of the approximation depends on the quadrilateral partition of the data. Strong variations in curvature or multiple fine details within a single surface lead to a higher approximation error. This behavior suggests that a partition with a limited amount of surface complexity per cell is desirable.

Figure 11 provides a comparison between an automatically generated quad-mesh and one that has been manually constructed. In both cases, each B-spline surface is defined by 6×6 control points. The manually constructed quad-mesh is designed in such a way that individual surfaces only contain curvature situations that can be handled easily. Even though the automatically generated quad-mesh has a higher number of surfaces, the quality of the B-spline approximation is still worse than the approximation based on the manually created quad-mesh.

Manually defining quad-meshes becomes impractical for large or complex objects. In this situation, an automatic solution is necessary. However, while our approximation algorithm works locally, established methods to compute quad-meshes work globally. Further research is necessary to deal with cases where additional restrictions arise, e.g., limited memory or incomplete data.

5 CONCLUSIONS AND FUTURE WORK

We have presented an algorithm for surface approximation that works locally. Given a point set and a quadrilateral base mesh, our algorithm computes a smooth B-spline surface model that closely approximates the input. The direct comparison shows that it offers a good trade-off between a purely C^0-continuous algorithm and an exact G^1-continuous scheme. We have demonstrated our method using nonlinear pointwise constraints. The approximation specific steps and continuity constraints can be exchanged by other algorithms. Our main contribution is the local approach, making it possible to locally use other approximation methods that usually could not be applied to point data due to data size or memory limitation.

In addition, we have discussed ways to analyze the resulting B-spline approximation quality. We can compute statistical measures and use them to trigger a re-computation with different parameters to improve an approximation. The colored surface quality visualizations are effective to ensure that desired approximation quality criteria are met.

Due to the locality of the algorithm, not all data need to be present in memory simultaneously. For each B-spline surface to be computed, the method only requires the data of the incident cells defined by the underlying quad-mesh topology. Thus, it is possible to merge computation results produced during different phases, enabling the processing of data sub-sets prior to progressing to other data parts and thereby minimizing expensive I/O operations.

Our algorithm does not rely on a specific choice of continuity constraints. Any G^1-continuous method can be used for the local surface construction. Our method utilizes B-splines of any order, and it could be generalized to arbitrary dimensions. Knot vectors do not necessarily need to consist of uniformly spaced knot values. However, for a quad-mesh of general topology, they must be equal for all B-spline surfaces and all dimensions.

While our research was motivated by constructing G^1-continuous surfaces, our method can be adapted to higher-order continuity. Locally, any G^k-continuous approximation approach can be employed. For G^k-continuity we must consider k rows/columns of G^k-relevant control points between individual processing phases.

We plan to extend the local nature of our approach to the entire processing pipeline, including data acquisition, initial meshing, and quad-mesh generation. Our method could then be applied to large data sets even on computers with limited memory. Further, it would open up the possibility to process parts of objects individually. For instance, when scanning a complex object too large to be scanned instantaneously, we could process individual, partial scans and still generate the desired overall approximation. In addition,

non-uniform rational B-splines (NURBS), see [23], provide additional degrees of freedom, i.e., control point weights, making it possible to improve overall approximation quality.

We also aim to combine local and global approaches. Currently, we compute G^1-relevant control points for all quad-cell corners and boundaries before we generate the inner control points. In a hybrid approach, it is possible to treat a cluster of cells together, computing G^1-relevant control points along the boundary of the cluster and internally enforcing continuity constraints with a global method. The challenge is to establish the cluster size based on determining when the computational cost of a global approach is still lower than that of a local approach. For this purpose, it will be necessary to explore computational complexity in much more detail, see Section 4.1.

Our processing pipeline can be executed in a fully automatic way. At this point, it is not yet optimal and additional expert knowledge can greatly improve approximation quality. As discussed in Section 2.2, a user can guide the generation of a high-quality quad-mesh. While many approaches for this purpose exist, there is still a lack of methods that compute high-quality quad-meshes suitable specifically for B-spline constructions. Especially for objects with edges, it is desirable to have cell boundaries lying on existing sharp edges and enforce tangent plane discontinuities in these locations.

ACKNOWLEDGEMENTS

This research was funded by the Fraunhofer High Performance Center for Simulation- and Software-Based Innovation and supported by the German Research Foundation (DFG) within the IRTG 2057 “Physical Modeling for Virtual Manufacturing Systems and Processes”, the German Federal Ministry of Education and Research through project 05M2013 (AniS), as well as a grant by the Deutsch-Französische Hochschule (DFH).

REFERENCES

[1] Hoschek, J., Lasser, D., and Schumaker, L. L., 1993. *Fundamentals of computer aided geometric design*. AK Peters, Ltd.

[2] Farin, G., 2002. *Curves and Surfaces for CAGD: A Practical Guide*, 5th ed. Morgan Kaufmann Publishers.

[3] Peters, J., and Fan, J., 2010. “On the complexity of smooth spline surfaces from quad meshes”. *Computer Aided Geometric Design,* **27**(1), pp. 96–105.

[4] Hahmann, S., Bonneau, G.-P., and Caramiaux, B., 2008. “Bicubic G^1 interpolation of irregular quad

meshes using a 4-split". In International Conference on Geometric Modeling and Processing, Springer, pp. 17–32.

[5] Bonneau, G.-P., and Hahmann, S., 2014. "Flexible G^1 interpolation of quad meshes". *Graphical Models,* **76**(6), pp. 669–681.

[6] Fan, J., and Peters, J., 2011. "Smooth bi-3 spline surfaces with fewest knots". *Computer-Aided Design,* **43**(2), pp. 180–187.

[7] Eck, M., and Hoppe, H., 1996. "Automatic reconstruction of B-spline surfaces of arbitrary topological type". In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, ACM, pp. 325–334.

[8] Peters, J., 1994. "Constructing C^1 surfaces of arbitrary topology using biquadratic and bicubic splines". In *Designing Fair Curves and Surfaces: Shape Quality in Geometric Modeling and Computer-Aided Design*. SIAM, pp. 277–293.

[9] Milroy, M. J., Bradley, C., Vickers, G. W., and Weir, D., 1995. "G^1 continuity of B-spline surface patches in reverse engineering". *Computer-Aided Design,* **27**(6), pp. 471–478.

[10] Lai, J.-Y., and Ueng, W.-D., 2001. "G^2 continuity for multiple surfaces fitting". *The International Journal of Advanced Manufacturing Technology,* **17**(8), pp. 575–585.

[11] Shi, X., Wang, T., and Yu, P., 2004. "A practical construction of G^1 smooth biquintic B-spline surfaces over arbitrary topology". *Computer-Aided Design,* **36**(5), pp. 413–424.

[12] Lin, H., Chen, W., and Bao, H., 2007. "Adaptive patch-based mesh fitting for reverse engineering". *Computer-Aided Design,* **39**(12), pp. 1134–1142.

[13] Lin, K.-Y., Huang, C.-Y., Lai, J.-Y., Tsai, Y.-C., and Ueng, W.-D., 2012. "Automatic reconstruction of B-spline surfaces with constrained boundaries". *Computers & Industrial Engineering,* **62**(1), pp. 226–244.

[14] Yoo, D.-J., 2011. "Three-dimensional surface reconstruction of human bone using a B-spline based interpolation approach". *Computer-Aided Design,* **43**(8), pp. 934–947.

[15] Zhao, X., Zhang, C., Xu, L., Yang, B., and Feng, Z., 2013. "IGA-based point cloud fitting using B-spline surfaces for reverse engineering". *Information Sciences,* **245**, pp. 276–289.

[16] Yoshihara, H., Yoshii, T., Shibutani, T., and Maekawa, T., 2012. "Topologically robust B-spline surface reconstruction from point clouds using level set methods and iterative geometric fitting algorithms". *Computer Aided Geometric Design,* **29**(7), pp. 422–434.

[17] Fan, J., and Peters, J., 2008. "On smooth bicubic surfaces from quad meshes". In International

Symposium on Visual Computing, Springer, pp. 87–96.

[18] Mourrain, B., Vidunas, R., and Villamizar, N., 2016. "Dimension and bases for geometrically continuous splines on surfaces of arbitrary topology". *Computer Aided Geometric Design,* **45**, pp. 108–133.

[19] Blidia, A., Mourrain, B., and Villamizar, N., 2017. "G^1-smooth splines on quad meshes with 4-split macro-patch elements". *Computer Aided Geometric Design,* **52**, pp. 106–125.

[20] Eck, M., DeRose, T., Duchamp, T., Hoppe, H., Lounsbery, M., and Stuetzle, W., 1995. "Multiresolution analysis of arbitrary meshes". In Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, ACM, pp. 173–182.

[21] The CGAL Project, 2019. CGAL user and reference manual. https://doc.cgal.org/4.14/Manual/packages.html. [Online; accessed 2019-10-14].

[22] Huang, J., Zhou, Y., Niessner, M., Shewchuk, J. R., and Guibas, L. J., 2018. "Quadriflow: A scalable and robust method for quadrangulation". In Computer Graphics Forum, Vol. 37, Wiley Online Library, pp. 147–160.

[23] Piegl, L., and Tiller, W., 2012. *The NURBS book*. Springer Science & Business Media.

[24] Rogers, D. F., and Fog, N., 1989. "Constrained B-spline curve and surface fitting". *Computer-Aided Design,* **21**(10), pp. 641–648.

[25] Hagen, H., and Schulze, G., 1987. "Automatic smoothing with geometric surface patches". *Computer Aided Geometric Design,* **4**(3), pp. 231–235.

[26] Kraft, D., 1988. "A software package for sequential quadratic programming". *Forschungsbericht-Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*.

[27] Johnson, S. G., 2014. The nlopt nonlinear-optimization package. http://github.com/stevengj/nlopt. [Online; accessed 2019-10-14].

LIST OF FIGURES

LIST OF TABLES