UiO : University of Oslo

Jostein Bratlie

Applications of blending splines in interactive geometric modeling

Thesis submitted for the degree of Philosophiae Doctor

Department of Informatics
Faculty of Mathematics and Natural Sciences

UiT – The Arctic University of Norway
R&D group Simulations – Narvik

2020

To my parents

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of *Philosophiae Doctor* at the University of Oslo. The research presented here is conducted under the supervision of professor Arne Lakså and co-supervision of professor Knut Mørken.

The thesis is a collection of five papers, presented in chronological order. The common theme is blending spline curves and surfaces and some of their applications in interactive geometric modeling. The papers are preceded by introductory chapters which relate the work together and provide motivation, shared theory, application examples, and concluding remarks and thoughts on future work.

This thesis work has been part of the *Dreamworld* project, which has been founded through the *research council of Norway*'s Verdikt program (no. 201511) [NFR09], where the main project participants are the Norwegian game studio, Funcom, and NUC. The project owner is the research group *Simulations* [UiT21] at UiT - The Arctic University of Norway - in Narvik (former Narvik University College – NUC). The main research and development (R&D) topics related to the project have been; *integration of social platforms in massive multiplayer online role playing game (MMORPG) worlds*, *easy access for all players*, *a seamless gaming world*, and *data representation and reduction*. The work included in this thesis falls under the latter.

I want to thank Funcom, UiO, and UiT Narvik, my supervisors, co-authors, co-workers, students, friends, family, compilers, and everyone who has been a part of this journey.

"*Thanks for all the fish!*", [Ada84].

Jostein Bratlie
Narvik, March 2020

List of Papers

Paper I

Bratlie, J. "Local refinement of GERBS surfaces with applications to interactive geometric modeling". In: *39th International conference applications of mathematics in engineering and economics AMEE13*. Ed. by Pasheva, V. and Venkov, G. Vol. 1570. AIP Conference Proceedings. AIP Publishing, 2013, pp. 18–25

Paper II

Bratlie, J., Dalmo, R., and Zanaty, P. "Fitting of Discrete Data with GERBS". in: *Large-Scale Scientific Computing 2013*. Ed. by Lirkov, I., Margenov, S., and Waśniewski, J. Vol. 8353. Lecture Notes in Computer Science. Springer, 2014, pp. 569–576

Paper III

Bratlie, J., Dalmo, R., and Bang, B. "Evaluation of smooth spline blending surfaces using GPU". in: *Curves and surfaces. 8th International Conference*. Ed. by Boissonnat, J.-D., Cohen, A., Gibaru, O., Gout, C., Lyche, T., Mazure, M.-L., and Schumaker, L. L. Vol. 9213. Lecture Notes in Computer Science. Springer, 2015, pp. 60–69

Paper IV

Bratlie, J. and Dalmo, R. "Exploring future C++ features within a geometric modeling context". In: *NIK: Norsk Informatikkonferanse*. Vol. 2019. 2019

Paper V

Bratlie, J. and Dalmo, R. "Blending spline polygon surface over arbitrary poly-mesh topology". Under revision.

The included papers are preprint versions in accordance with the individual publishing agreements.

Contents

List of Figures

List of Acronyms

API application programming interface

B-function blending function

CPU central processing unit

CUDA compute unified device architecture

DirectX Microsoft® DirectX®

ERB expo-rational blending function

ERBS expo-rational B-spline

GBC generalized barycentric coordinate

GERBS generalized expo-rational B-spline

GPGPU general-purpose GPU

GPU graphics processing unit

IGA isogeometric analysis

IGM interactive geometric modeling

LERB logistic expo-rational basis function

OpenCL open computing language

OpenGL open graphics library

R&D research and development

UiT Narvik UiT - The Arctic University of Norway - in Narvik

Chapter 1

Introduction

"*Virtual worlds are expected to be more and more realistic. With increased artistic freedom and demand for realism the amount of data describing these worlds increase with each released title. Especially with respect to MMORPGs. For each released title the expectations of realism is heightened. With increased realism better visual effects, increased resolution of models and more stunning scenarios are expected. In addition, the quality of story-telling, dialogs, voice work and acting is expected to increase similarly. Moreover, this demands a richer animation base for in-game, non cinematic characters. More importantly this results in a larger data footprint and a more intricate production pipeline as different tools supports different data representations*", [Bra11].

1.1 Research topics and objectives

The primary focus of the research group, *Simulations* [UiT21a], over the last two decades, has been to develop the theory of blending splines and to explore their applications. The overall goal of the ph.d. work has been to investigate some of these application areas with a particular focus on interactive geometric modeling (IGM) and video game technology applications.

The central hypothesis of this ph.d. work is based on

- the possible applications of a combined set of unique properties, enjoyed by blending spline constructions, Chapter 2,
- the idea that these make them suitable for various applications of IGM, and
- moreover, this seems to be a good match for applications to animation-related video-game assets.

The combined set of unique properties include, but are not limited to, minimal support, controllable smoothness, vanishing derivatives and Hermite interpolation properties of the basis functions, and local self-contained geometric coefficients in the form of parametric objects, such as curve-, surface-, polygon- and volume-constructions.

1.1.1 Application to video-game technology

R&D in the field of blending splines has mainly focused on constructional properties, basis functions, and applications to finite element methods. A limited number of efforts had been focused on applications to interactive and free form

geometric modeling. The three most notable are Lakså's ph.d.-thesis on exporational B-spline (ERBS) [Lak07], Andresen's m.sc.-thesis on user interfaces for free-form artistic ERBS-modeling [And08], and Rasmussen's m.sc.-thesis on tensor-product blending spline volume construction [Ras06]. Besides introducing the ERBS, and its basic properties, Lakså's thesis also makes the case of the construction's favorable connection to free form modeling by introducing a set of standard local geometry construction types and a set of local/global geometry combinations. On the other hand, Andresen's m.sc.-thesis centers around different 3D gadgets and how they are best applied for editing purposes in a blending spline surface context. The thesis also touches on the use of double-knots to construct parametric warps in the structures, which again leads to controllable geometric discontinuities (splits and holes) and blending spline surfaces as local surfaces for hierarchical modeling. Moreover, this is the underlying technique we used for the application example in Section 5.6. Rasmussen's m.sc.-thesis applies a tensor-product blending spline volume construction on smoothed discrete data, where the construction is exemplified through visualization of fire explosions.

We identified three main video-game-related areas where we wanted to apply the construction: animation tracks, skinning, and interactive modeling. In Section 5.3 and Section 5.4, the application efforts made to skinning and animation tracks are covered. The sections cite two application papers, [HBD14] and [BD14], where the former was submitted as a part of Haavardsholm's m.sc. thesis.

1.1.2 Interactive geometric modeling

IGM is a topic that is covered throughout the thesis. In Paper III, we connect the blending spline surface construction with modern graphics application programming interfaces (APIs). We presented a rendering method that was constructed by exploiting this connection through custom software pipelines. This resulted in a fixed and stable performance cost. Moreover, this enables IGM through disjoint and parallelizable tasks (such as rendering and editing), resulting in responsiveness for the end-user, e.g., 3D-model artist. In Paper I, we propose two local refinement schemes for use with blending spline surfaces. These are founded on the *minimal support* and *vanishing derivatives* properties of the blending spline basis function. This enables local refinement, not by knot insertion, but by exploiting the hierarchical spatial nature of the blending spline construction. Additionally, the tensor-product irregular grid and arbitrary poly-mesh constructions, which are the primary focus of [BBD16] and Paper V, respectively, loosens the topological constraints which come with tensor-product constructions and gives 3D modeling artists more freedom to express form.

1.1.3 GPU and GPGPU utilization

With the introduction of programmable graphics processing units (GPUs) and thereof enabling of the individual programmable stages, such as the tessellation-control and -evaluation shaders, blending spline evaluators can be moved to

the GPU in its entirety and, as an effect, offload the central processing unit (CPU). In Paper III, we present a rendering method for blending splines where we exploit the similarities between the blending splines and the linear B-spline, presented by Lakså in [Lak14]. We were able to construct a render method exploiting a standard API feature of the graphics API to render each blending spline surface patch individually. This means that the evaluation of a complete surface construction can be parallelized over the GPU grid. As the modern general-purpose GPU (GPGPU) APIs becomes more streamlined, the same evaluation techniques used for accurate rendering could be used for accurate evaluation. The technique was developed for tensor-product surfaces; however, it can quite easily be extended to polygon patch domains, which is used in Paper V.

1.1.4 Arbitrary poly-mesh representations

One of the main goals of free-form spline-based surface modeling is its extension into the utilization of arbitrary topology. This is one of the factors which propelled the development of constructions such as T-splines [Sed+03]. Moreover, this has shown to be challenging for blending spline constructions, especially when maintaining minimal support and support for arbitrary interchangeable local geometry. Lakså touches on the main reason for this issue, in [Lak07], the challenge of finding a parametric curve that is mapped continuously across the inner boundaries of the global construction (referred to as defining constant parameter lines). Then, in [Lak13], a construction which trades stiffness in the global construction and local geometry spanning further than the first neighborhood, for the ability to span a tensor-product surface over irregular grids, was proposed. In Paper V this restriction is lifted as we suggest a construction utilizing polygon constructions for local surface representation. We achieve this by moving a step away from an explicit parametric construction. In return, we have to pay the cost of projecting the discrete boundary coefficients of the embedded polygon back onto the parametric space. This to limit and find the parametric domain for which we can compute and utilize generalized barycentric coordinates (GBCs) as an underlying implicit parameterization.

1.1.5 Reparameterization techniques

Reparameterization is a central technique used in blending spline constructions. The local sub-geometry of each local geometry construction must be reparameterized to match the parameter domain of a current blending spline knot interval under evaluation. Two application approaches have been investigated:

- local refinement by reparameterization, Paper I, and
- sub-objects used for reparameterization into local geometry, either to represent a sub-domain of the surface or some projection or intersection, Papers III and V, and Chapter 5.

1.1.6 Type-safe programming

As one extends the blending spline construction, the construction itself quickly extends into a dimensional problem where the practical implementation is a matter of indexing some data structure or domain which is based on relational, hierarchical, and dimensional metadata. The prototype work for Paper III and [BBD16] led to a need to solution this challenge. The solution manifested as a desire to exploit existing programming tools, such as the compiler, to verify and limit the programmable construction of geometric constructions. By classifying the properties of a given construction and grouping them in compile-time and runtime categories, we wanted to utilize the static type system of modern C++ so that the compiler could limit how we build geometric construction. In Paper IV, the grouped subset of properties are linked with concrete C++ techniques and utilized to construct a prototype library that uses the static type-system to restrict the language when prototyping geometric constructions. The accompanied prototype is included in appendix A.

1.2 Research methodology

Our research methodology is based on a brainstorming process followed by discussions among research group members. A hypothesis is formed, and verification is sought, primarily through mathematical proofs and software prototyping. The process is then repeated until the theory is proven or falsified. After a proven prototype has been analyzed and edge cases of more complex paths have been explored, the results are compiled into a final paper.

We emphasize verification through rigorous software development; therefore, all concrete theories, theorems, definitions, constructions, and algorithms, are verified through software prototypes before being submitted for publishing. Due to the custom nature of all our software prototypes, this is a time-consuming process. However, it provides a concrete counter-part to the mental exercise of mathematical proofs. This proof verifying method is one of the strongest motivations for utilizing tools such as the ones described in Paper IV.

Throughout the project, we have investigated edge cases represented by numerous small experimental projects developed through a variation on extreme programming. The thesis work represents hundreds of small prototypes written in a variety of programming languages. A selection of results from such experiments has ended up in this thesis through the included publications.

The backbone of our software production process is our in-house library suite, GMlib, which is described in Chapter 4. The features developed and implemented in the library are primarily driven by the collective work conducted by the *Simulations* R&D group. The library suite should be classified as an experimental library due to the high nature of R&D material. Quality assurance (Q&A) of the library is conducted, as a natural part of the teaching methodology, through the usage of the library, e.g., m.sc. students use the library as a development tool for research-driven project assignments. Since 2008 the author of this thesis has held the role of curator and maintainer of GMlib and its support

software and is to date the primary developer of the new prototype version, GMlib2, and its support software.

1.3 List of contributions

Paper I [Bra13]: Local refinement of GERBS surfaces with applications to interactive geometric modeling

This paper investigates knot-insertion on blending spline surfaces for the purpose of modeling added local features. This requires the refinement technique to provide an exact construction after refinement. A proof is provided, which shows that an exact constructional refinement does not yield the exact same mathematical form. It provides two strategies for layered local refinement where the construction is exact and on the same given form.

1. A *multi-layered* method: exact refinement is achieved by introducing multiple inner knots where each knot is associated with its own refinement patch and where the individual parametric domains are shifted.
2. A *multi-level* method: exact refinement is achieved by introducing a single inner knot on an existing knot interval where the inner knot is associated with a blending spline refinement patch identical over that knot interval. Multi-level blending is achieved by applying the refinement technique iteratively on the new refinement patches.

Paper II [BDZ14]: Fitting of discrete data with GERBS

This paper investigates the compression of discrete univariate data using approximation and interpolation techniques through blending spline constructions. Two results, in particular, are provided.

3. Benchmarking of two partitioning techniques: curvature and inflexion. The inflexion based techniques are of particular interest as this can be used for finding good points of interest for animation track data, as in [BD14], which is discussed in Section 5.4.
4. The spline-based approach performs at least as well as a least-squares approach on smooth input data and has improved stability for oscillating irregular input data.

Paper III [BDB15a]: Evaluation of smooth spline blending surfaces using GPU

This paper was motivated by an identified connection between the particular blending spline surface construction presented in [Lak13] and the nature of the tessellation shader API of modern graphics hardware. The patch type introduced with modern rendering APIs, designed for simple B-spline and Bézier patches, has overlapping properties with a blending spline patch (knot interval).

5. This paper proposes a concept describing an organizational scheme for the rendering of blending spline surface constructions exploiting the tessellation shader API introduced with modern GPU hardware.

6. The proposed scheme acts as an abstraction layer and a concept for building and organizing the relevant shaders. The scheme should be transparent to all well-established rendering techniques, regardless of whether they be raster-, ray-, path-based, or otherwise.

Paper IV [BD19]: Exploring future C++ features within a geometric modeling context

This paper is a bi-product of the prototype work conducted for Papers III and V. The paper investigates how to utilize current and coming C++ features to improve implementation safety when prototyping abstract geometric modeling constructions. It focuses on restricting the error space through type-safe and static compile-time programming rather than post-implementation regression tests. These language techniques were crucial in developing the prototype software accompanying Paper V.

7. In addition to describing a set of implementation techniques, it provides a minimal sample library, API, and example, listed in appendix A.

Paper V [BD21]: Blending spline polygon surface over arbitrary poly-mesh topology

This paper is the main contribution to the thesis. The paper investigates a blending spline surface construction over arbitrary poly-mesh topology using local surfaces suitable for artistic free-form geometric modeling. The most significant results include.

8. A smooth surface construction over arbitrary poly-mesh topology by blending local polygon surface constructions of Bézier-type.

9. The requirements and limitations of the construction.

10. A method for implicitly generating explicit spline knot vectors from a spline knot graph.

11. A method for transitively controlling and defining smooth parameter lines and curves across internal edges on the resulting spline surface. These are used, for instance, to define continuous directional derivatives.

1.4 Remarks and future work

1. The thesis and included papers have been written utilizing the TEX Live [Rah+21] LATEX suite (pdfLatex) and a custom set of CMake build scripts for automation. Throughout the thesis and included papers, the illustrations included are either graphics produced with custom software or illustrations and plots generated by the PGF/TikZ package, tkz-euclid, tikz-uml, and PGFPlots.

2. The custom prototyping software that is written to produce the graphical illustrations and the numerical and structural experiments for the various papers have been developed using an in-house software suite based on C++. The various experiments have been produced over a significant time period utilizing this software suite; see the first part of Chapter 4 for details.

3. All the custom software applications and libraries used for prototyping included in the thesis have in majority been written and developed by the thesis author, except;

   - GMlib (version 1): which has been in development since 1994, by various contributors; mainly Arne Lakså and Børre Bang. However, the thesis author has been its maintainer since 2008.
   - Paper II: the majority of the numerical experiments were written by Peter Zanaty.
   - Section 5.3: the skinning experiment was in its completeness implemented by Birgitte Haavardsholm.
   - Section 5.6: the hole-construction experiment was implemented by Aleksander Pedersen.

4. The utilization of blending spline methods for applications related to video games is intriguing. This thesis only touches on a couple of application areas; however, future work on applications into animation track control and objects with shared local geometry is of particular interest.

5. The blending splines enjoy a set of properties which in combination makes them versatile in applications to free-form modeling. In particular, they provide an artist with new and exciting degrees of freedom and detailed control. However, the degrees of freedom are potential drawbacks as the constructions grow in complexity and computational cost. Research into efficient solutions for given application scenarios, such as complex world and character modeling, should be one of the next steps forward.

6. Continuing the previous remark, the ability to interchange both the underlying blending functions (B-functions) and the local geometry types in the same construction is an undeniable positive degree of freedom. This lets the user control both the parameterization, continuity, computational cost, mathematical space, embedding properties, and more. Research efforts into best practices should be prioritized, from an artistic modeling perspective, as this can be overwhelming when applying the construction to a problem.

Paper I

7. Interchanging the B-function and, in turn, changing the basis function of the blending spline surface changes the reparameterization of the construction locally. This, in turn, can affect various visual and computational components of the construction, e.g., the slope towards a parametric knot. When a surface is initially refined by an artist, the surface should be preserved exactly upon refinement and no unwanted geometric artifacts should be introduced. For instance, the refinement can be performed on a blending spline construction where the basis function has been modified in order to introduce wanted properties and artifacts; therefore it is crucial to preserve the construction itself upon refinement.

Paper II

8. The inflexion method for defining points of interest along a discrete curve can be combined with other methods for blending spline curve control, such as

   a) manually marking points of interest,

   b) sharing coefficients among local geometry,

   c) applying constraints such as minimizing rotation frames in the construction, and more.

   Moreover, as this can enable customizable, controllable, and efficient animation track curve constructions, it is an exciting topic for future research.

Paper III

9. The published paper used additional nomenclature for blending spline knot grids, knots and patches, namely render lattices, render locus and render block, respectively. The purpose of this was to distinguish the construction purely as a rendering method. The nomenclature has been excluded from the preprint included in the thesis, as it only adds confusion.

10. Constructing rendering-pipelines for this construction is complex. Especially when utilizing a combination of programming languages, platforms, and technologies. A possible future research effort could be developing a domain-specific language that could compile source code, binary modules, and rendering pipelines from strict differential geometry descriptions. An example of a similar technology is [Hu+19a; Hu+19b] utilized for differential programming of physical simulations.

Paper IV

11. The paper provides a constructional method for prototyping geometric objects using modern C++ features while utilizing the compiler as a strict verification tool. A natural extension of the paper is to provide a set of well-defined concept types for such constructions, and a thorough property-type categorization, e.g., compile-time or runtime-dependent types.

12. The next step is then to experiment with reflection methods. This has the potential to enable directly controllable definitions of complex differential geometry concepts in code with little effort and great payoff.

Paper V

13. Besides a broader investigation into application possibilities, the paper lists two primary topics for future work; efficient methods for interactive editing and utilization of different methods for computation of generalized barycentric coordinates.

    a) For interactive editing, the evaluation methods used when reevaluating a modified partition need to be as efficient as possible. This is due to the nature of the parametric domains, see Chapter 3, which are associated with polygon constructions.

    b) Exploring generalized barycentric coordinates has not been an active part of the research but has been utilized more like a utility in the form of mean value coordinates. Even though the mean value coordinates are underperforming for non-convex domains, it has been a stable method during the initial R&D of the construction. However, future research should focus on finding a feasible replacement, which is both directly evaluable and strictly positive. One such approach is a solution based on power coordinates, [Bud+16], which shows promising results and seems like a good starting point.

14. Utilizing local sub-surfaces as a middle-layer is a natural part of this construction. Exploring different types of local geometry for different purposes would be an interesting avenue for future research. Local geometry can differ from basic polygon constructions to embedded curves in the form of sub-object construction in external geometric objects, e.g., surfaces [Lak07, Section 6.6] or volume-embedded surfaces, such as the constructions described in Section 4.3.

1.5 Outline of additional research results

The following is a list of additional contributions to papers and research projects made during the ph.d.-period. These have been omitted to keep in line with the research objectives of the thesis and to streamline the exposition.

1. Bratlie, J. “Methods for userguided compression algorithms”. In: *NIK: Norsk Informatikkonferanse*. Vol. 2011. 2011

2. Dechevsky, L. T., Bratlie, J., and Gundersen, J. “Index Mapping between Tensor-Product Wavelet Bases of Different Number of Variables, and Computing Multivariate Orthogonal Discrete Wavelet Transforms on Graphics Processing Units”. In: *Large-Scale Scientific Computing 2011*. Ed. by Lirkov, I., Margenov, S., and Waśniewski, J. Vol. 7116. Lecture Notes in Computer Science. Springer, 2012, pp. 402–410

3. Dechevsky, L. T., Bratlie, J., Bang, B., Lakså, A., and Gundersen, J. “Wavelet-based lossless one- and two-dimensional representation of multidimensional geometric data”. In: *37th International conference applications of mathematics in engineering and economics AMEE11*. Ed. by Venkov, G., Kovacheva, R., and Pasheva, V. Vol. 1410. AIP Conference Proceedings 1. AIP Publishing, 2011, pp. 83–97

4. Dechevsky, L. T., Bratlie, J., and Gundersen, J. “Index Mapping between Tensor-Product Wavelet Bases of Different Number of Variables, and Computing Multivariate Orthogonal Discrete Wavelet Transforms on Graphics Processing Units”. In: *Large-Scale Scientific Computing 2011*. Ed. by Lirkov, I., Margenov, S., and Waśniewski, J. Vol. 7116. Lecture Notes in Computer Science. Springer, 2012, pp. 402–410

5. Dalmo, R. and Bratlie, J. “Discrete Wavelet Compression of ERBS”. in: *Large-Scale Scientific Computing 2013*. Ed. by Lirkov, I., Margenov, S., and Waśniewski, J. Vol. 8353. Lecture Notes in Computer Science. Springer, 2014, pp. 577–584

6. Dalmo, R. and Bratlie, J. “Data approximation using a blending type spline construction”. In: *40th International conference applications of mathematics in engineering and economics AMEE14*. Ed. by Pasheva, V. and Venkov, G. Vol. 1631. AIP Conference Proceedings. AIP Publishing, 2014, pp. 147–152

7. Bratlie, J. and Dalmo, R. “Motion capture data represented using a blending type spline construction”. In: *40th International conference applications of mathematics in engineering and economics AMEE14*. Ed. by Pasheva, V. and Venkov, G. Vol. 1631. AIP Conference Proceedings. AIP Publishing, 2014, pp. 153–157

8. Dalmo, R., Bratlie, J., Bang, B., and Lakså, A. “Smooth spline blending surface approximation over a triangulated irregular network”. In: *International Journal of Applied Mathematics* vol. 27, no. 1 (2014), pp. 109–119

9. Dalmo, R., Bratlie, J., and Zanaty, P. “Image processing with LERBS”. in: *ICNPAA 2014 World Congress: 10th International conference on Mathematical Problems in Engineering, Aerospace and Sciences*. Ed. by Sivasundaram, S. Vol. 1637. AIP Conference Proceedings. AIP Publishing, 2014, pp. 271–278

10. Haavardsholm, B., Bratlie, J., and Dalmo, R. “Surface deformation over flexible joints using spline blending techniques”. In: *ICNPAA 2014 World Congress: 10th International conference on Mathematical Problems in Engineering, Aerospace and Sciences*. Ed. by Sivasundaram, S. Vol. 1637. AIP Conference Proceedings. AIP Publishing, 2014, pp. 377–383

11. Bratlie, J., Dalmo, R., and Bang, B. “Wavelet compression of spline coefficients”. In: *Numerical Methods and Applications 2014*. Ed. by Dimov, I., Fidanova, S., and Lirkov, I. Vol. 8962. Lecture Notes in Computer Science. Springer, 2015, pp. 246–253

12. Dalmo, R., Bratlie, J., and Bang, B. “Performance of a wavelet shrinking method”. In: *Numerical Methods and Applications 2014*. Ed. by Dimov, I., Fidanova, S., and Lirkov, I. Vol. 8962. Lecture Notes in Computer Science. Springer, 2015, pp. 262–270

13. Pedersen, A., Bratlie, J., and Dalmo, R. “Spline representation of connected surfaces with custom-shaped holes”. In: *Large-Scale Scientific Computing 2015*. Ed. by Lirkov, I., Margenov, S., and Waśniewski, J. Vol. 9374. Lecture Notes in Computer Science. Springer, 2015, pp. 394–400

14. Dalmo, R., Bratlie, J., and Zanaty, P. “Image compression using an adjustable basis function”. In: *Mathematics in Engineering, Science and Aerospace* vol. 6, no. 1 (2015), pp. 25–34

1.6 Organization of the thesis

The thesis is organized into two main parts. The first part contains introductory chapters which provide vital background information and common theory shared by the papers. The second part is organized into chapters (papers) which contain the included research papers. The papers have been included as preprint editions of the original publications, where editions have been made to streamline notation, illustrations, and more. Both the bibliography of the introductory part as well as the bibliographies of each of the papers are included immediately after each respective chapter.

1.7 Note on included papers and shared theory sections

The theory forming the basis of the research project is the R&D into blending spline constructions. During the ph.d.-period the general blending spline R&D has introduced various constructions which again have been used in the included

papers: Papers I to III and V. Therefore, to unify the reading experience, the blending spline theory has been extracted from the papers and collected in Chapter 2, and the individual papers refer back to Chapter 2. Additionally, Paper III is a direct precursor to Paper V, and the wording and nomenclature of the prior has been updated to reflect the current state of the dependent theory. See Section V.6.2, in particular, for spline- and geometry notation.

Chapter 2

Blending splines

2.1 Patchwork surface representations

The terminology *patchwork* can be used to describe composite surfaces: surfaces that are composed of piecewise surface patches. Composite surfaces were first developed in the 1960s by C. de Boor [de 62] and J. Ferguson [Fer64] in parallel.

The most popular patch constructions are built either from the tensor-product framework, such as in the case of Bézier patches, or the more general B-spline and NURBS patches, or from boundary curve approaches, such as Coons' patches [Coo67] and Gregory patches [Gre74]. A significant part of the thesis work is centered around polygonal structures. Patchworks built with polygonal patches are usually constructed using S-patches [LD89] or the recent general Bézier patches [SV18] and Gregory-style S-patches [HK18]. Even though polygonal patch constructions are flexible with respect to topology, they are inherently more complex due to their non-explicit parametric spaces.

2.2 Blending construction representations

Patchwork constructions are single-layered with respect to geometric spaces. An alternative CAGD approach is blending constructions (or manifold constructions). Blending constructions are multi-layered, where global geometry is generated by blending, evaluations of, local geometry. These types of surface constructions can be grouped into two branches; those which expand the parametric space and those which expand the geometric space.

In the first branch, we find Coons' patches and their generalizations, including Gordon type surfaces [Gor69a; Gor69b], which elevates boundary curves or curve networks into a surface via blending.

The other branch of surfaces utilizes blending functions to blend local geometry of equal parametric dimension into global geometry, e.g., from local surfaces to a global surface patch. Hartmann's G^n blending techniques for curves and surfaces [Har01], where sub-partitions of local geometry are used to construct smooth geometry partitions, seems to be the first instance of this type. Circle spline curves [SLY05] are another early approach, where local circle partitions are blended by linear functions to create G^2- and C^2-continuous curves.

2.2.1 Manifold blending versus spline blending

Later, two new families of blending constructions were developed; the manifold-based surface blending approach [YZ04] and the ERBS [DLB06]. These methods

blend overlapping parametric local surfaces to form global surface patches, which again form a continuous global surface. This can be seen in contrast to the patchwork methods, where patches are instead stitched together along their adjacent boundaries in order to form continuous surfaces. The two families of blending constructions; manifold-based and blending splines, respectively, are fairly similar. The significant differences are their theoretical background, approach to topological control, parameter space control, and local geometry choices.

The manifold-based construction was recently upgraded and improved [TZ11] to support piecewise-smooth boundaries. Furthermore, isogeometric analysis (IGA) methods [MC17] and polygonal mesh constructions [ZL17] based on manifold basis functions have emerged.

The ERBS construction was generalized as generalized expo-rational B-spline (GERBS) in [DBL09], and later specialized, e.g., NUERBS [DLB04] or BFBS [Dec10]. Research on blending splines has mainly focused on basic properties of the construction itself, where the most notable are the associated basis-functions [DBL09; DLB06; Lak07], and on applications to simulations and numerical approximation with respect to finite element methods. The thesis of Zanaty is a good starting point [Zan14], as well as the newly published application to IGA using ERBS triangles [Kra19; Kra20].

2.2.2 Patchwork vs. Blending constructions

The main challenges of the patchwork constructions, besides developing a surface patch with a given internal smoothness, are to control the smoothness across the boundary shared by adjacent patches and to deal with the twist compatibility problem which arises at the patch corners. These challenges have to be addressed for each individual patch in the composition. However, the twist compatibility problem vanishes due to the blending construction's vanishing derivatives property (in relation to the Hermite order property $\mathcal{P}5$) which is reiterated in Section 2.6.1.

2.3 B-functions

Throughout the thesis, now dated variations and subsets of the blending spline theory have been utilized. The reason is the parallel advances in R&D into these methods of geometry construction. To ease the reading experience, the blending spline related material has been chosen to be presented in a uniformly. The blending spline notation and terminology forms the basis for the interpretation leading to the main thesis results; therefore, this has been chosen as the nomenclature and terminology for the thesis itself.

With the introduction of the blending spline terminology, a notation for non-polynomial variations of the blending splines emerged [Lak14]. This is based on a two-function approach [Lak13; LB15] where the GERBS is formulated by adjusting the recursive B-spline definition. In the following, this approach is reiterated.

2.3.1 B-spline function

Consider a parametric B-spline function

$$f(t) = \sum_{i=1}^{n} c_i b_{d,i}(t),$$

for $t \in [t_d, t_n]$, where c_i are the local coefficients (control points), $b_{d,i}$ are the B-splines defined over the knot vectors $\{t_i\}_{i=0}^{n+d}$, and n and d are the number of basis functions and polynomial degree, respectively. Following, the B-spline basis function over the knots is defined recursively as follows

$$b_{d,k}(t) = w_{d,k}(t) b_{d-1,k}(t) + (1 - w_{d,k+1}(t))\, b_{d-1,k+1}(t), \tag{2.1}$$

where

$$w_{d,i}(t) = \frac{t - t_i}{t_{i+d} - t_i}, \tag{2.2}$$

and the recursion terminates with

$$b_{0,i}(t) = \begin{cases} 1; & \text{if } t_i \leq t < t_{i+1} \\ 0; & \text{otherwise} \end{cases}, \qquad i = k, \ldots, k+d.$$

2.3.2 B-function

A symmetric blending function (B-function) is a smooth monotonous permutation function that satisfies the following properties

$\mathcal{P}1$: $\mathfrak{B}(t) : [0,1] \mapsto [0,1]$,

$\mathcal{P}2$: $\mathfrak{B}(t)$ has fixed end points: $\mathfrak{B}(0) = 0$ and $\mathfrak{B}(1) = 1$,

$\mathcal{P}3$: is continuous and monotone,

$\mathcal{P}4$: is point-symmetric: $\mathfrak{B}(1-t) = 1 - \mathfrak{B}(t)$, and

$\mathcal{P}5$: has an Hermite order $S \geq 0 : \mathfrak{B}^{(j)}(0) = \mathfrak{B}^{(j)}(1) = 0,\ j = 1, \ldots, S$.

Property $\mathcal{P}4$ is optional and specifies point symmetry around the point $(0.5, 0.5)$. The order of the B-function is determined by how many times the function can be differentiated and still keep its Hermite order (property $\mathcal{P}5$). Some examples of B-functions are

- linear $\mathfrak{B}(t) = t$,
- trigonometric $\mathfrak{B}(t) = sin^2(\frac{\pi}{2}t)$,
- polynomial $\mathfrak{B}(t) = 3t^2 - 2t^3$, and
- rational $\mathfrak{B}(t) = \frac{t^3}{t^3 + (1-t)^3}$,

with respective Hermite order (S): 0, 1, 1 and 2. The example B-functions are visualized in Figure 2.1.

Remark 2.3.1. Symmetric B-functions are presumed throughout the thesis.

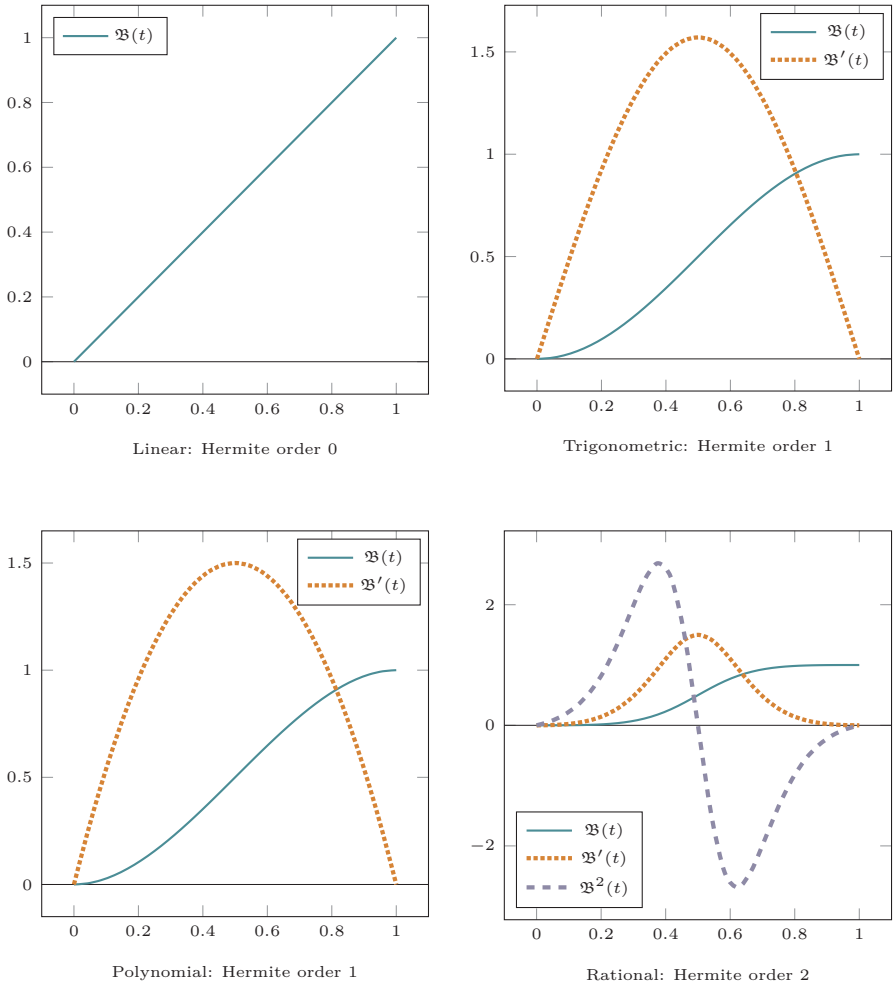


Figure 2.1: Plots of example B-functions and their Hermite-order derivatives. The B-functions are listed in Section 2.3.2. The derivatives of the rational B-function are scaled down by factors of 2 and 5, respectively.

2.3.3 Specialized blending functions

The first specialized B-function, expo-rational blending function (ERB), was introduced with the ERBS in [DLB06]. The ERB has a Hermite order, $S = \infty$. In [Lak07, p. 2.5] the scalable subset was introduced with the purpose of, in addition, being an efficient blending spline basis-function for applications of IGM. Written in the form of a B-function it is expressed as follows

$$\mathfrak{B}(t) = 1.6571 \int_0^t \exp\left(-\frac{(t-\frac{1}{t})^2}{t(1-t)}\right)\, dt.$$

The scaleable subset also has a Hermite order, $S = \infty$.

Several more B-functions have been developed since the scalable subset; most notably, the logistic expo-rational basis function (LERB) and Fabius approximations, illustrated in Figure 2.2.

The LERB was introduced with the logistic expo-rational B-splines (LERBS) by Dechevsky and Zanaty in [DZ13, section 2.2]. Its default set [DZ13, equation 19], can be expressed as a B-function as follows

$$\mathfrak{B}(t) = \frac{1}{1+\exp\left(\frac{1}{t}-\frac{1}{1-t}\right)}.$$

The LERB also has a Hermite order, $S = \infty$.

Schemas to construct blending functions approximating the Fabius function, presented in [Olo19], are among the latest additions to the families of B-functions. With the approximation schema the Hermite order can be designed. The two primary variations are the ones introduced with [Olo19, equation 8 and 21], expressed as follows

$$\mathfrak{B}(t) = \frac{1}{2} - (\frac{9}{16}\cos(\pi t) - \frac{1}{16}\cos(3\pi t)), \tag{2.3}$$

and

$$\mathfrak{B}(t) = t - \frac{\sin(2\pi t)}{2\pi}, \tag{2.4}$$

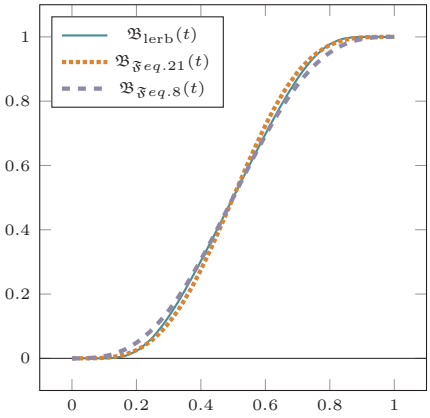
respectively. These Fabius approximations, (2.3) and (2.4), have Hermite order $S = 3$ and $S = 2$, respectively.

2.3.3.1 Application and intend use

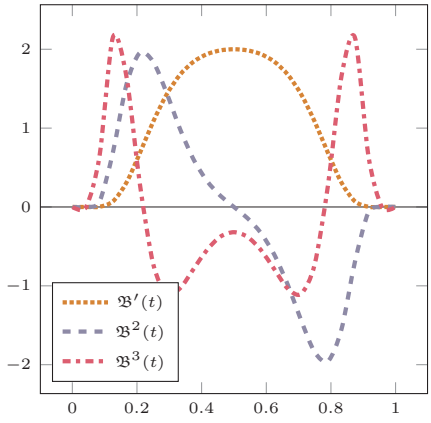
The B-functions included in this chapter are suitable for different applications in IGM.

ERB
: applications to IGM, primarily evaluation on CPU [DLB06].

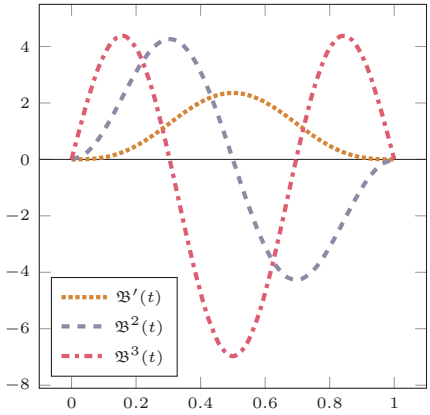
LERB
: applications to IGM, primarily evaluation in CPU/GPU kernel thread space [DZ13].



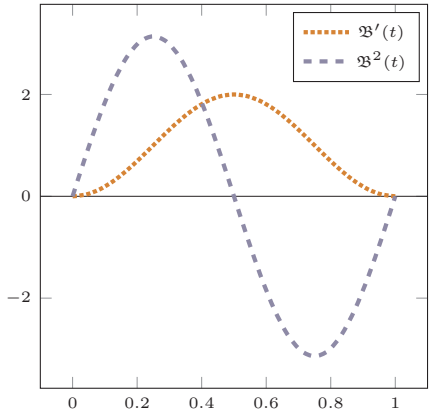
LERB and Fabius plots



LERB: Hermite order ∞



Fabius (2.3): Hermite order 3



Fabius (2.4): Hermite order 2

Figure 2.2: Plots of three specialized B-functions (top left) and their Hermite-order derivatives. Derivatives: all Hermite-order derivatives are plotted except for the LERB (Hermite-order ∞) where the three first are plotted. Scaling; factors of 5 and 50 respectively are used to scale down the LERB second and third derivatives, the Fabius equation 8 approximation (2.3) second and third derivatives are respectively scaled down by factors of 2 and 10, and the Fabius equation 21 approximation (2.4) second derivative is scaled down by 2.

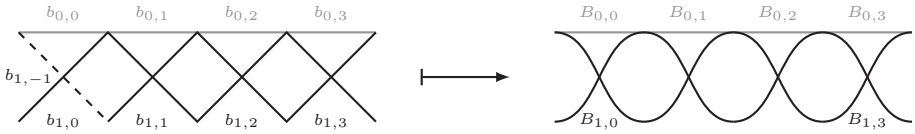


Figure 2.3: Redistributed B-spline basis function as a blending spline basis function using a smooth B-function.

Fabius approximations
: fast computation in CPU/GPU kernels for a given Hermite-order [Olo19] (customizable approximation scheme for constructing Fabius approximating B-functions).

A couple of other relevant B-functions include the very first B-function, by Ying and Zorin [YZ04]

$$\mathfrak{B}(t) = \frac{\exp\left(\frac{-2}{t}\exp\left(\frac{-1}{1-t}\right)\right)}{\exp\left(\frac{-2}{t}\exp\left(\frac{-1}{1-t}\right)\right) + \exp\left(\frac{-2}{1-t}\exp\left(\frac{-1}{t}\right)\right)},$$

which was based on a non-point-symmetric function by Navau and Garicia [NG00] combined with Hartmann's rational construction [Har01], and the Beta-function basis function (BFB) [Dec10] used in blending spline triangle-based finite element methods [Zan14].

2.4 Blending spline basis

Now that the B-function has been defined, the recursive B-spline basis function can be reformulated, and we get the formula for a general B-spline basis as follows

$$B_{d,k}(t) = \mathfrak{B} \circ w_{d,k}(t) B_{d-1,k}(t) + (1 - \mathfrak{B} \circ w_{d,k+1}(t)) B_{d-1,k+1}(t), \qquad (2.5)$$

where $\mathfrak{B}(t)$ is a B-function, $w_{d,k}$ is defined as in (2.2), and the recursion terminates analogously to (2.1). The second order general B-spline basis ($d = 1$) is known as a *blending spline basis*, see Remark 2.4.1 and Remark 2.4.4. Figure 2.3 illustrates the effect of the first degree reformulation.

Remark 2.4.1. The GERBS basis or *blending spline basis* construction is equivalent to a second-order general B-spline ($d = 1$), (2.5), where $\mathfrak{B}$ is a B-function and the local coefficients depend on the parametric parameter, i.e., t.

Remark 2.4.2. The blending spline basis is utilized throughout the thesis work; therefore, we will assume $d = 1$ and hence $B_{d=1,k}(t) = B_k(t)$ for the remainder of this document.

Remark 2.4.3. The blending spline basis is referred to as the ERBS basis when redistributed with the ERB, referred to as the LERBS basis when redistributed with the LERB, and so on.

Remark 2.4.4. General B-splines is not to be confused with the generalized B-splines [CLM05; MPS11], which is a similar but different generalization of classical B-splines. The main focus of generalized B-spline applications is IGA [MPS11].

2.4.1 Properties and two-function blending

A blending spline function can then be formulated as follows

$$\begin{aligned} f(t) &= \sum_{i=1}^{n} \ell_i(t) B_{d=1,i}(t), \\ &= \sum_{i=1}^{n} \ell_i(t) B_i(t), \end{aligned} \tag{2.6}$$

for $t \in [t_d = 1, t_n)$, where $\ell_i(t)$ is a vector of local curves, $B_{d=1,i} = B_i$ are the blending spline basis, and $\{t_i\}_{i=0}^{n+1}$ is an increasing knot vector.

Continuing, [Lak14, theorem 3] states the following properties of blending spline constructions, besides the properties inherited from polynomial B-splines

$\mathcal{G}1$: the continuity is extended with the order of the B-function,

$\mathcal{G}2$: the speed of the general B-splines is affected by order of the B-function.

These two properties enable the user to choose a B-function based on the needs given by a specific application problem.

In [Lak14] Hermite-order properties are demonstrated utilizing blending of two functions. Blending of two functions is the core of the blending splines, i.e., represented on one knot interval of the spline. The expanded formulae look as follows

$$\begin{aligned} f(t) &= (1 - \mathfrak{B}(t))\, \ell_1(t) + \mathfrak{B}(t)\, \ell_2(t) \\ &= \ell_1(t) + (\ell_2(t) - \ell_1(t))\ \mathfrak{B}(t), \end{aligned}$$

where f is the global function and ℓ_1 and ℓ_2 are the local functions at the start and end, respectively. The general formula for derivatives then follows as follows

$$f^{(j)}(t) = \ell_1^{(j)}(t) + \sum_{i=0}^{j} \binom{j}{i} \mathfrak{B}^{(i)}(t)(\ell_2(t) - \ell_1(t))^{(j-i)}(t).$$

Lemma 1 [Lak14] states that the global function f interpolates the first local function, ℓ_1, in the start and the last local function, ℓ_2, in the end with position and derivatives up to order S_1 and S_2, respectively ($S_1 = S_2$ when using symmetric B-functions)

$$\begin{aligned} f^{(j)}(0) &= \ell_1^{(j)}(0), \quad j = 0, 1, \ldots, S_1, \\ f^{(j)}(1) &= \ell_2^{(j)}(1), \quad j = 0, 1, \ldots, S_2. \end{aligned}$$

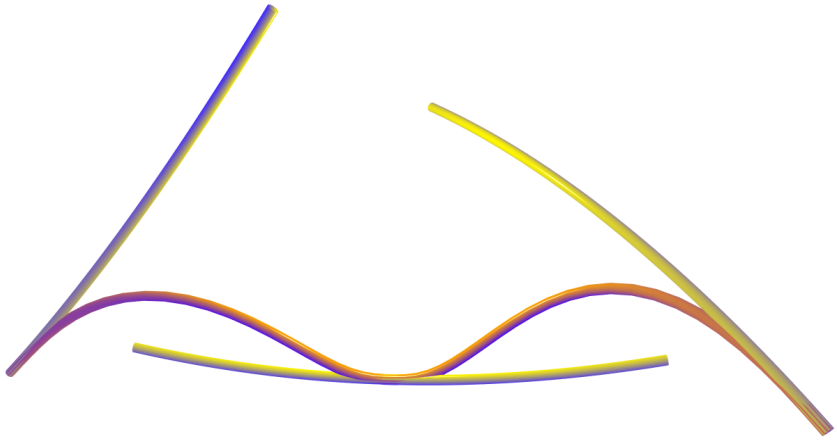


Figure 2.4: Parametric blending spline curve constructed from three local quadratic Bézier curves, using a Fabius approximated B-function.

Furthermore lemma 2 [Lak14] states that if the local functions are equal in the start or end, i.e. $\ell_1(0) = \ell_2(0)$ and $\ell_1(1) = \ell_2(1)$, respectively, the Hermite interpolation increases by 1, respectively:

$$f^{(j)}(0) = \ell_1^{(j)}(0), \quad j = 0, 1, \ldots, S_1 + 1,$$
$$f^{(j)}(1) = \ell_2^{(j)}(1), \quad j = 0, 1, \ldots, S_2 + 1.$$

2.5 Curves

In one-dimensional (1D) parametric space, a tensor-product blending spline construction is interpreted as a curve. The curve can be defined similarly to a function as follows

$$C(u) = \sum_{i=1}^{n} \bar{C}_i(u)\ B_i(u), \tag{2.7}$$

for $u \in (d, n]$, where $\bar{C}_i(u)$ is a vector of local curves, B_i are the blending spline basis, and $\{u_i\}_{i=0}^{n+1}$ is an increasing knot vector. An example is visualized in Figure 2.4, where a blending spline curve is constructed from three local Bézier curves.

2.6 Tensor-product surfaces

Continuing from the curve definition, 1D, in (2.7), the formula for a blending spline surface, 2D, can be expressed in the same manner.

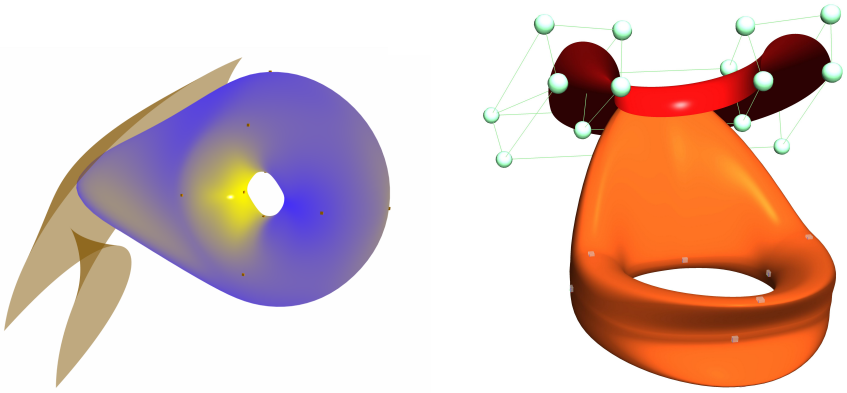


Figure 2.5: Two examples of modified blending spline surfaces. Both surfaces started out as approximations of a torus using 4x4 local Bézier surfaces, where the left and right surfaces utilized quadratic and cubic Bézier surfaces, respectively.

A tensor-product blending surface is defined as follows

$$S(u,v) = \sum_{i=1}^{n_u} \sum_{j=1}^{n_v} \bar{L}_{i,j}(u,v)\ B_j(v)\ B_i(u), \tag{2.8}$$

for $u \in [u_{d_u}, u_{n_u})$ and $v \in [v_{d_v}, v_{n_v})$, where $\bar{L}_{i,j}(u,v)$ is a net of local surface, B_i and B_j are the blending spline basis functions, and $\{u_i\}_{i=0}^{n_u+1}$ and $\{v_j\}_{j=0}^{n_v+1}$ are increasing knot vectors. Two examples of edited blending splines surfaces are visualized in Figure 2.5.

2.6.1 Blending spline surface patchwork interpretation

One consequence of the B-function property $\mathcal{P}5$, given by the blending spline property $\mathcal{G}1$, for a function $f(t) = \sum_i \ell_i(t) B_i(t)$ is that $f'(t_i) = \ell_i'(t_i)$. Analogously, for a tensor product surface (2.8), we obtain

$$\frac{\partial}{\partial u} S(u_i, v) = \frac{\partial}{\partial u} \bar{L}_{i,j}(u_i, v) B_j(v) + \frac{\partial}{\partial u} \bar{L}_{i+1,j}(u_i, v) B_{j+1}(v),$$

and

$$\frac{\partial}{\partial v} S(u, v_i) = \frac{\partial}{\partial v} \bar{L}_{i,j}(u, v_i) B_i(u) + \frac{\partial}{\partial v} \bar{L}_{i,j+1}(u, v_i) B_{i+1}(u),$$

at any knot u_i or v_j, respectively, for $v_j \le v < v_{j+1}$ or $u_i \le u < u_{i+1}$.

Using this property, a tensor product blending spline surface construction with $d_u = d_v = 1$ (Remark 2.4.2) can be interpreted as a surface patchwork by considering the sums (2.8) from i to $i+1$ and j to $j+1$ as individual patches. For example, the patch obtained for $i = 2, j = 3$ is adjacent to the patch with $i = 3, j = 3$ along the knot line at $j = 3$. The rectangular domain of the tensor

product knot grid facilitates continuous and smooth parameter lines between adjacent surface patches. In contrast to traditional patchworks, there are no twist compatibility problems at the knots due to the *minimal support* and the *vanishing derivatives* of the basis functions.

2.7 Beyond tensor-product curves and surfaces

The blending spline constructions can be extended to higher-order differentials such as volumes and flows. Tensor-product volume constructions were first explored in [Ras06]. The master thesis explored a tensor-product volumetric extension of blending splines where it was applied to smoothed scalar data visualization and exemplified by visualizing fire explosions.

Furthermore, Dechevsky and Zanaty have explored more complex surface constructions used for various applications with respect to radial basis approximation and finite-element methods. A good literary starting point is the thesis of Zanaty, [Zan14]. Recently, Kravetc has explored triangle-based blending spline constructions for finite-element-based IGA in [Kra19; Kra20].

The papers included in the thesis utilizes different constructions and B-functions

Paper I explores surfaces and presumes a given B-function,

Paper II explores functions and utilized the classic ERB,

Paper III explores tensor-product patches and bases their implementation on the LERB, and

Paper V explores GBC polygon constructions and utilizes a blending spline basis based on a B-function Fabius appoximation; [Olo19][eq. 21].

Chapter 3

Blending spline polygon surfaces

The blending spline surface construction presented in Paper V is constructed under a set of constructional invariants.

- The surface construction is constructible over arbitrary poly-mesh topology,
- it is a patch-based construction, i.e., adaptable to a rendering-scheme utilizing the graphics API's patch-type primitives, such as in Paper III, and
- it utilizes well-known, de facto editing techniques for local surfaces, i.e., Bézier-type local surfaces and affine transformations.

Additionally, it inherits all requirements implicitly imposed by any blending spline surface construction and tries to enforce an explicit parameterization approach.

The work builds on the construction first explored by Lakså in [Lak07, section 6], where ERBS-triangles were introduced over a homogeneous barycentric coordinate parametrization. Our proposed construction extends the triangle-based construction into polygon-constructions over arbitrary poly-mesh topologies. Following this extension, the local parametrization changes to generalized barycentric coordinates (we utilize mean value coordinates), and as the local polygon-constructions must support non-convex parametric domains, we utilize ribbon-based generalized Bézier patches [SV18], as local geometry. However, this means we are moving a step away from a complete explicit parameterization approach.

3.1 Arbitrary poly-mesh topology

In [ZL17] surface approximation of arbitrary poly-mesh topology by applying manifold-based surfaces was explored using Catmull-Clark subdivision surfaces as local geometry functions. Their method was successful; however, they point out that parametric control is challenging, and further investigations are necessary. In Paper V, we apply the blending spline to arbitrary poly-mesh topology, Where we address parametric control by using the blending spline approach and utilizing a cross-boundary tensor-product approach.

3.2 The underlying issue

In [Lak07, section 6.8] Lakså identifies the triangle-based construction's central challenge as the issue of constructing a parametric coordinate line across the

internal boundary of the blending spline patchwork, for which one can produce continuous derivatives. Lakså identifies this as the issue of constant parameter lines.

To obtain a global construction with controllable continuity, when blending local surfaces to form patches, we need to produce a parameterization that can produce a continuous parameter line across patch boundaries.

Paper V proposes a solution based on an approach first proposed in [VRS11], for non-blending polygon patchworks, which utilizes tensor-product parameterization across edges boundaries of two adjacent polygon surfaces. With the hierarchical blending spline construction, we also need to control the parameter space explicitly. We do this by introducing an internal polygon-based patchwork in the parameter space of each individual local surface.

Experience has show that to see and understand the underlying issue; it helps to know the steps of a solution. Given two adjacent patch covers on an arbitrary poly-mesh, as visualized on the left in Figure 3.3, $\hat{Q}^0$ and $\hat{Q}^1$. Furthermore, given one of the patches, we seek a "common" parameterization between the parametric domain of that patch and a local surface, such that we can define a parameter line across the common boundary of the adjacent patches. We construct our solution in the following three steps

- partition the parameter domain of the local surface into sub-domains of the patch-topology of the global construction, see Figure 3.4 and Sections V.7.4 and V.8.2,
- introduce side-based parametrization to the GBC parameterized local sub-surface domains, see top of Figure 3.5 and Section V.6.4.3, and
- control the parameterization of a cross-boundary parameter line, which augments the side-based parameters, by mapping the parameter line through a polygon patchwork embedded in the local surface parameter domain, see bottom of Figure 3.5 and Section V.8.3.

For blending purposes, we are using tensor-product parametrization across the adjacent patch boundaries; therefore, a comparison to a tensor-product variation is used to highlight the issue in the following.

Figure 3.1 illustrates parameterizations for both a tensor-product and a polygon surface construction, where the tensor-product surface parameter domain is embedded in the same implicit space as a polygon surface construction. The embedding of the tensor-product parametrization into the underlying space is trivial. However, due to the nature of GBCs and its bijective mapping properties [Jac13; SHF13; Sch17], it is not.

3.3 Local surface parameter domain patchwork

An approach used to solve the cross-boundary parameter line issue for polygon surface patchworks was proposed in [VRS11]. It is utilized for patchwork applications of their subsequent constructions [SV18; VSK16; VSK17]. The

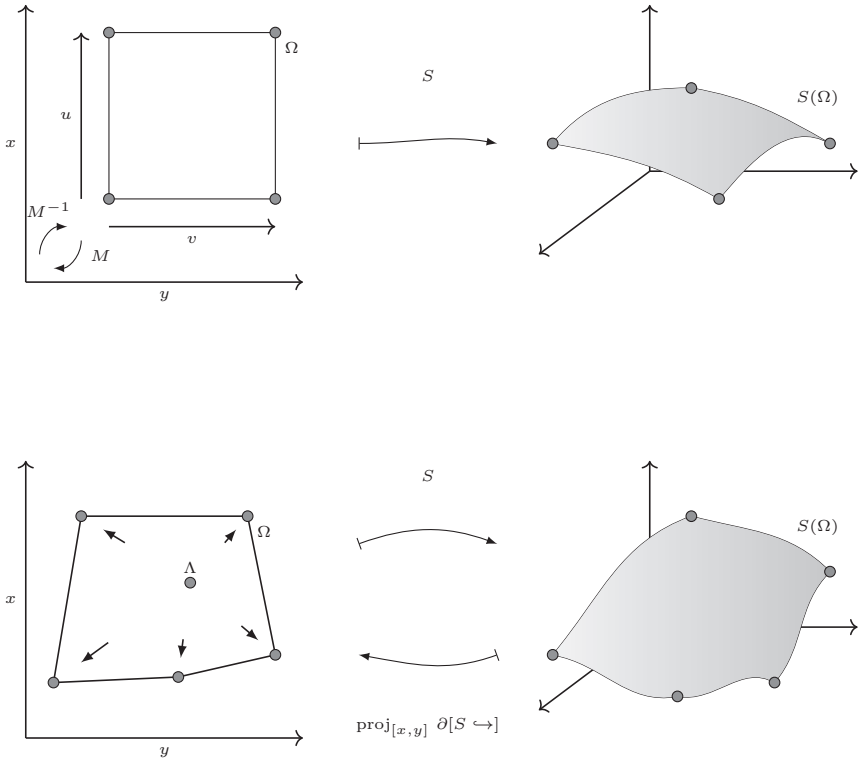


Figure 3.1: Parametric surface mapping. Top: tensor-product surface, where M represents the mapping between the $[x, y] \subset \mathbb{R}^2$ and $[u, v] \subset \mathbb{R}^2$ spaces. Bottom: GBC polygon surface. $\partial[S \hookrightarrow]$ refers to the embedded boundary coefficients of S.

approach enables a continuous parameterization across adjacent polygon-shaped parameterizations when applied to the local surface parameter domain for the patch-tessellated topology. The process is illustrated in Figure V.5. Furthermore, this enables us to describe the same local parametric direction from both sides of an adjoining shared boundary and define continuous derivatives.

When blending sub-partitions of local polygon surfaces, as we do when constructing blending spline patches, the local parameter line must be mapped between the parameterizations of all the local sub-polygons. Figure 3.2 illustrates the continuous cross-boundary mapping of a constant parameter line between the tensor-product parametrization of a tensor-product blending spline surface and a local tensor-product patch and Figure 3.3 demonstrates the discontinuous mapping between the blending spline polygon surface and one of the local polygon surface constructions.

The second point illustrated through Figures 3.2 and 3.3 is that the shape of

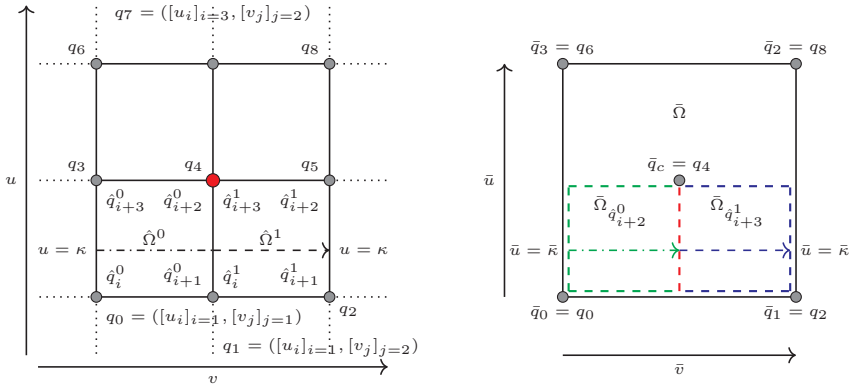


Figure 3.2: Direct mapping of a parameter line, κ, on a tensor-product blending spline surface. Left: the parametric space of the tensor-product blending spline surface. Each q_i corresponds to a pair of intersecting knots from the $[u]$ and $[v]$ knot vectors. Right: the parametric space of the local tensor-product surface associated with the knot at q_4.

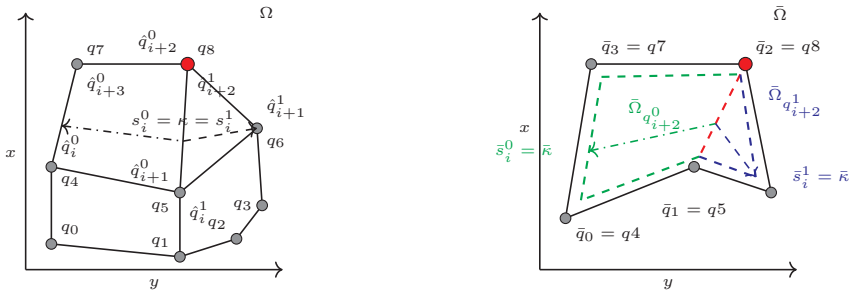


Figure 3.3: Direct mapping of a parameter line, κ, on a blending spline poly-mesh topology. Left: a minimal poly-mesh topology. Right: the local polygon surface associated with knot q_8.

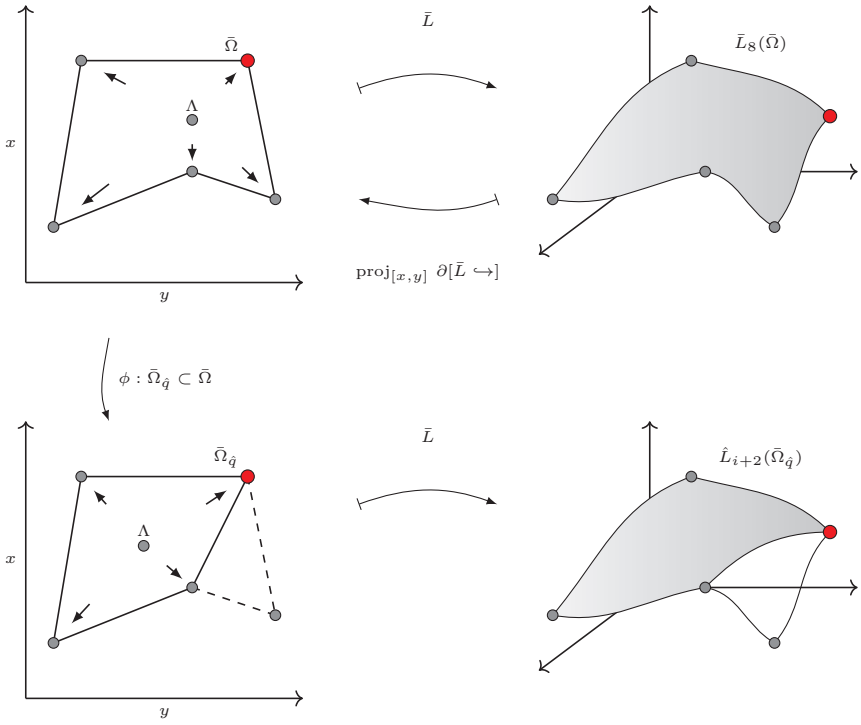


Figure 3.4: GBC blending spline polygon surface and sub-surface mapping relationship. Visualizes the mapping relationship between a blending spline polygon surface and a sub-surface in that surface. Where $\partial[\bar{L} \hookrightarrow]$ refers to the embedded boundary coefficients of $\bar{L}$, and ϕ induces the local parametric sub-domain.

the overlapping parametric domains of the polygon construction differs, contrary to the tensor-product case.

In Figure 3.4, we can see that the shape of a single local sub-surface, used for blending over one blending spline patch, is determined by the embedding of the local sub-surfaces' parametric domain. Comparing the shape of the resulting local sub-polygon domain to the shape of the blending spline patch domain, $\hat{\Omega}^0$, from the left figure of Figure 3.3, we see that this shape is not completely congruent to $\bar{\Omega}_{q^0_{i+2}}$. This situation is easy to worsen if we apply a transformation to the space of the local surface.

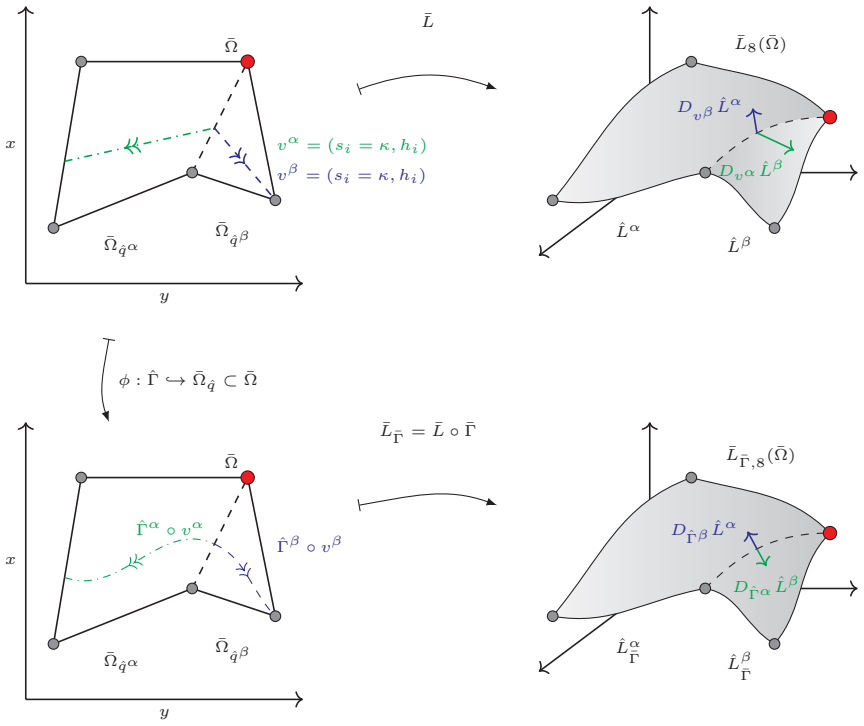


Figure 3.5: Continuing from Figure 3.4. Shows the effect of the mapping solution from Paper V, when fixing a side-parameter s_i on two adjacent sub-surfaces.

3.4 Controlling the parameter space

To control the discontinuity, we propose to embed a patchwork in the parameter space of the local polygon surface, which follows the tessellation given by the global patch topology through the local sub-surfaces. The continuity of the parametric space is then controlled and limited by the chosen polygon construction used in the patchwork. This is illustrated in Figure 3.5.

For every blending spline patch: by overlapping the given blending spline polygon patch's polygon domain with each of its associated local sub-surface polygon domains, we get congruent polygon domains, Section V.10.1. We can now find continuous parameter lines across adjoining patch boundaries using the method described in Paper V. See the paper for the continuation.

3.5 Interactive geometric modeling

The parametric domain of the polygon construction is a projection of the boundary coefficients from the polygons embedded space back onto its parametric

space, as shown in the bottom of Figure 3.1. Moreover, this mapping is affected differently depending on whether this is a patch, local surface, or local sub-surface mapping.

Patch $\hat{\Omega}$; produced from a projection of the patch's boundary coefficients and is the local sub-surface's embedded interpolation points. These are primarily affected by affine transformations to the local surfaces.

Local surface $\bar{\Omega}$; produced from a projection of the local polygon construction's embedded boundary coefficients. These are primarily affected by local editing of the boundary coefficients themselves.

Local sub-surface $\bar{\Omega}_{\hat{q}}$; produced as a sub-domain of $\bar{\Omega}$. It is Primarily affected by changes to $\bar{\Omega}$ and the local projection of the interpolation point on the local surface, given that the local surface is an internal coefficient construction to the blending spline surface.

Chapter 4

Prototyping differential geometry

This chapter is included to demonstrate how our in-house library, GMlib2, and the principles it is built on, presented in Paper IV, are utilized to prototype sub-object constructions for patchwork-centric blending spline constructions.

4.1 GMlib

Our realized prototypes, developed as part of our research methodology, utilize a set of in-house application suites named GMlib. These come in two versions, GMlib and GMlib2.

GMlib (version $\leq$ 0.7) [LBK06]

GMlib, version "one" of the software suite, has been in continuous development since 1994. This library is primarily written in C++98, with various upgrades to support a selection of C++11 functionality. Furthermore, it uses template meta-programming to specify types and dimension of containers and parameterized objects generically. The part relevant for the current topic is modeled using well-known object-oriented paradigms, such as inheritance and virtual member functions. Additionally, the library has its own linear algebra core, application-specific containers, a scenegraph structure, an open graphics library (OpenGL) [KHR2-] abstraction layer with a visualization framework for parametric objects, a triangulation system framework with functionality supporting finite element method (FEM) analysis applications, and more. The surrounding support software is based on the Qt application framework[The19]. It is utilized for R&D and teaching efforts at UiT Narvik.

GMlib2 (latest version as of May 14, 2021) [UiT21b]

The prototyping work for the new blending spline theory has shown us that one of the major challenges and keys to efficient prototyping are consistent and safe handling of indexation into data and objects across different data containers, object-hierarchies, and spatial as well as logical dimensions. When these structures reach a given complexity, the indexation rules needed for consistency can only be classified as index-magic and are, *in all applications*, subject to human errors.

During the prototyping work for Papers III and V([BD21; BDB15a]), we started developing the next version of GMlib, a suitable C++ R&D library which distinguishes between information known at compile-time and runtime, where the primary approach is to utilize the compile-time template-language of C++ fully. Version 2 of the software suite was mainly written to support the research

and development of these efforts. This version also uses the Qt application framework for everything application, UX and graphics related. The library is written to support the newest C++ standard (currently C++17 going onwards to C++20). It utilizes the Blaze C++ [Igl+12] linear algebra abstraction library for linear algebra, vector- and matrix representations. For mesh-loading and topology structures it uses the Openmesh [Bot+02] library.

The primary application area of GMlib2 is prototyping of differential geometry constructions, objects are organized hierarchically, and virtual data structures are shared across both objects and mathematical spaces. The library is, as with GMlib, used in both R&D and for teaching purposes. Paper IV was written to document some of the key approaches and the techniques we are utilizing in the development of GMlib2. Furthermore, the paper points to a set of future language improvements that will improve type-safety and source-code size footprints.

4.2 Parametric objects

Let us take a look at how to construct a parametric surface. We exemplify this with the definition of a torus. The purpose is to give an example of the C++ syntax used when defining a parametric object. Further examples can be found in GMlib2. Moreover, its general design principles are described in Paper IV.

A parametric torus class constructed in GMlib2 is declared as follows

```
template <typename SpaceObjectEmbedBase_T
          = ProjectiveSpaceObject<spaces::D3R3SpaceInfo<double>>,
          template <typename> typename PObjEvalCtrl_T
          = evaluationctrl::SurfaceEvalCtrl>
class Torus : public Surface<SpaceObjectEmbedBase_T, PObjEvalCtrl_T> {
public:
  using Base = Surface<SpaceObjectEmbedBase_T, PObjEvalCtrl_T>;

  GM2_DEFINE_DEFAULT_PARAMETRIC_OBJECT_TYPES

  // Constructor(s)
  template <typename... Ts>
  Torus(Unit wheelradius, Unit tuberadius1, Unit tuberadius2,
        Ts&&... ts)
    : Base(std::forward<Ts>(ts)...), m_wheelradius{wheelradius},
      m_tuberadius1{tuberadius1}, m_tuberadius2{tuberadius2}
  {
  }

  // Members
  Unit m_wheelradius;
  Unit m_tuberadius1;
  Unit m_tuberadius2;

  // PSurf interface
  PSpaceBoolArray isClosed() const override;
  PSpacePoint     startParameters() const override;
  PSpacePoint     endParameters() const override;

protected:
  EvaluationResult evaluate(
    const PSpacePoint& par, const PSpaceSizeArray& no_der,
    const PSpaceBoolArray& from_left
    = utils::initStaticContainer<PSpaceBoolArray, PSpaceVectorDim>(
      true)) const override;
};
```

All parametric object sub-classes inherit the parametric object class, which in turn inherits the embedding object space class. For a parametric tensor-product surface, the parametric object class is represented through the `Surface` using-directive. This is a using directive that partially specializes a `ParametricObject` class to a two-dimensional parametric space. Furthermore it needs two boilerplate declarations. The `Base` type-definition and the `GM2_DEFINE_DEFAULT_PARAMETRIC_OBJECT_TYPES` macro. These are needed for consistent declarations of common types utilized for sub-object type-inheritance and API consistency.

The type-safe vulnerability posed by the common types macro will disappear with the upcoming C++20 concepts, and the need for the boilerplate code itself will disappear with reflection-support and meta-programming scheduled for C++23 and beyond.

Next, we need to reimplement four member functions answering questions about non-static functionality.

`isClosed` – defines whether the parametric object is mathematically closed in a parametric direction, e.g., a circle is closed, a line is open, and an object, such as a Bézier or blending spline curve, can be both depending on the embedding of its coefficients.

`startParamters`/`endParameters` – the start and end parameter space values, e.g., a circle would have the parametric domain $[0, 2\pi)$, while a Bézier curve would have $[0, 1]$, and a torus would be defined on the domain $[0, 2\pi) \times [0, 2\pi)$.

`evaluate` – evaluates the object at a given position for a set of given parameters.

A torus surface evaluator can be implemented as follows

```
template <typename SpaceObjectEmbedBase_T,
          template <typename> typename PObjEvalCtrl_T>
typename Torus<SpaceObjectEmbedBase_T, PObjEvalCtrl_T>::EvaluationResult
Torus<SpaceObjectEmbedBase_T, PObjEvalCtrl_T>::evaluate(
  const PSpacePoint& par, const PSpaceSizeArray& no_der,
  const PSpaceBoolArray& /*from_left*/) const
{
  // Extract parameter values
  auto const& u        = par[0];
  auto const& v        = par[1];
  auto const& no_der_u = no_der[0];
  auto const& no_der_v = no_der[1];

  // Helpers
  auto const sv   = m_tuberadius2 * sin(v);
  auto const cv   = m_tuberadius2 * cos(v);
  auto const cu   = (m_wheelradius + m_tuberadius1 * cos(v)) * cos(u);
  auto const su   = (m_wheelradius + m_tuberadius1 * cos(v)) * sin(u);
  auto const cusv = m_tuberadius1 * cos(u) * sin(v);
  auto const susv = m_tuberadius1 * sin(u) * sin(v);

  /* Return value container */
  EvaluationResult p(no_der_u + 1, no_der_v + 1);

  /* Evaluate the surface; position and first derivatives in u and v */
  p(0, 0) = {cu, su, sv, 1};                          // S
  if (no_der_u > 0) p(1, 0) = {-su, cu, 0, 0};        // Su
  if (no_der_v > 0) p(0, 1) = {-cusv, -susv, cv, 0};  // Sv

  return p;
}
```

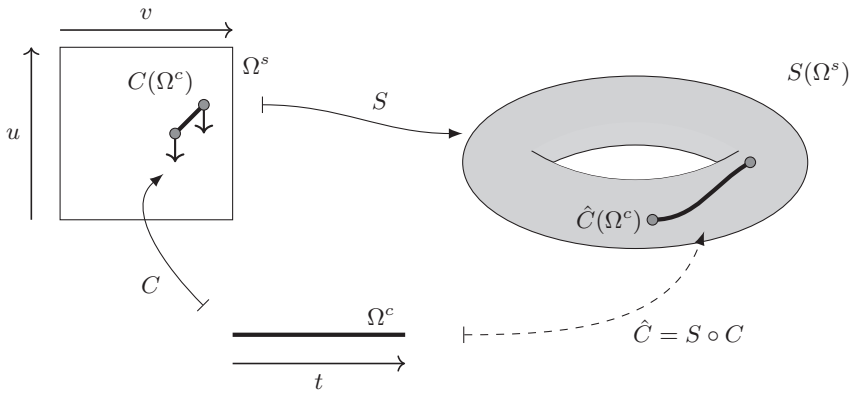


Figure 4.1: A sketch of parametric Hermite curve (1D) embedded in the parametric space of a torus (2D).

4.3 Parametric sub-objects

Let us consider the following code snippet showing how to construct an embedded object

```
1  // Namespace shortening
2  namespace gm2  = gmlib2;
3  namespace gm2p = gm2::parametric;
4
5  // Construct helper types
6  using R3           = gm2::spaces::D3R3SpaceInfo<double>;
7  using ProjSpaceObj = gm2::ProjectiveSpaceObject<R3>;
8  using Torus        = gm2p::Torus<ProjSpaceObj>;
9  using SubCurve = gm2p::SubCurveInSurface<gm2p::HermiteCurveP2V2, Torus,
10                                          ProjSpaceObj>;
11 using SCPoint  = SubCurve::PSpaceObject_Point;
12 using SCVector = SubCurve::PSpaceObject_Vector;
13
14 // Torus
15 auto torus = make_unique<Torus>(3.0, 1.0, 1.0);
16
17 // SubCurve in torus
18 auto const subcurve_p0 = SCPoint{0.0, 0.0};
19 auto const subcurve_p1 = SCPoint{2 * M_PI, 1.5 * M_PI};
20 auto const subcurve_v0 = SCVector{20.0, 0.0};
21 auto const subcurve_v1 = SCVector{2.0, 0.0};
22 auto       subcurve    = make_unique<SubCurve>(
23   torus.get(), subcurve_p0, subcurve_p1, subcurve_v0, subcurve_v1);
```

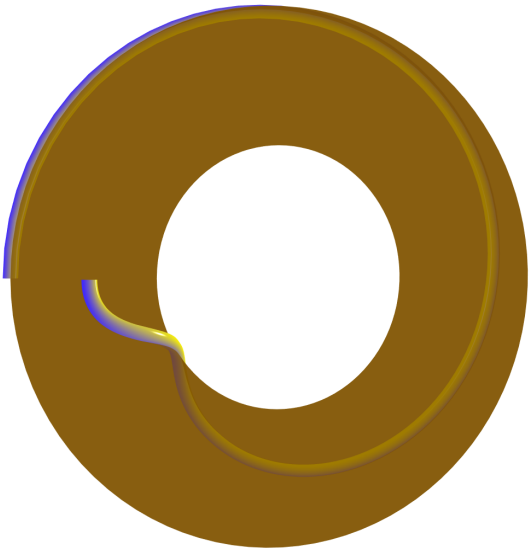


Figure 4.2: A parametric Hermite curve (1D) embedded in the parametric space of a torus (2D).

This code first constructs a torus (2D) embedded in a 3D space (`ProjectiveSpaceObject`), (line 15), where the embedding space info, $\mathbb{R}^3$, is specified on (line 6). The torus is constructed with a wheel radius of 3[units] and inner and outer tube-radii, both equal to 1. Next, we construct a Hermite curve (1D) embedded in the torus' parametric space (2D). The Hermite curve interpolates two positional parameters (start and end), $p_0 = (0.0, 0.0)$ and $p_1 = (2\pi, \frac{3\pi}{2})$, respectively, in the torus' parametric space, $p_0, p_1 \in [(0, 0), (2\pi, 2\pi)]$. In the same manner, it defines its start- and end tangents, v_0 and v_1. The data footprint constructed by these two objects consists of their coefficients and spatial matrices. The remaining data, making up the structure of the objects, are optimized away by the compiler at compile-time, e.g., dimension, parametric domain, object types, and other structural information. This information can be stored on demand in, for instance, a static `constexpr` class-variable if need be. A differential sketch of the object construction is visualized in Figure 4.1, and a rendering of the object is visualized in Figure 4.2.

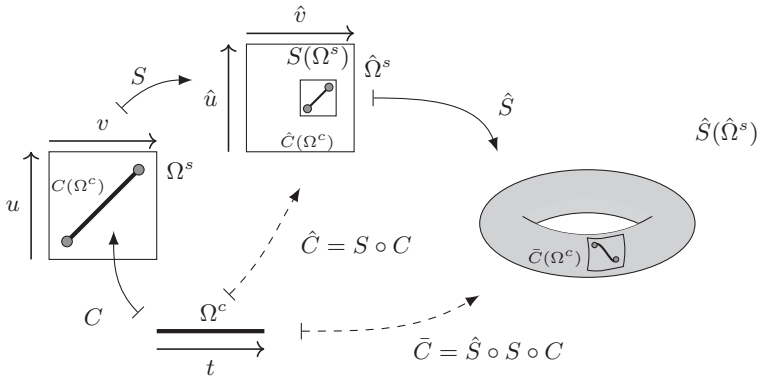


Figure 4.3: A sketch of a parametric line (1D) embedded in the parametric space of a plane (2D) which again is embedded in the parametric space of a torus (2D).

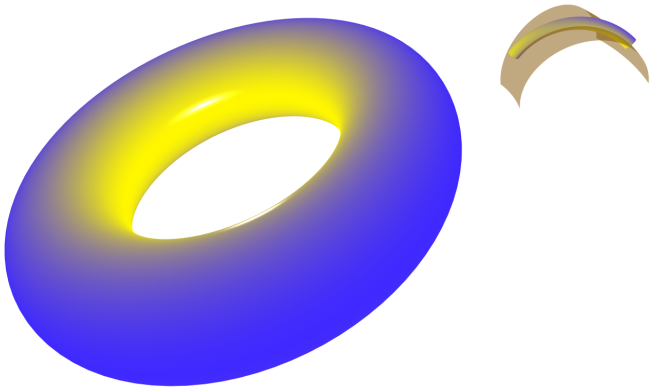


Figure 4.4: A parametric line (1D) embedded in the parametric space of a plane (2D) which again is embedded in the parametric space of a torus (2D).

4.3.1 Two levels of embedding

C++' type-system deduces the structural information of the embedded object from their embedding objects at compile time, therefore the static structure from Section 4.2 can be hierarchically chained, and a multilevel embedding can be defined as visualized in Figures 4.3 and 4.4. The source code for this example is included in Appendix B.1.2.

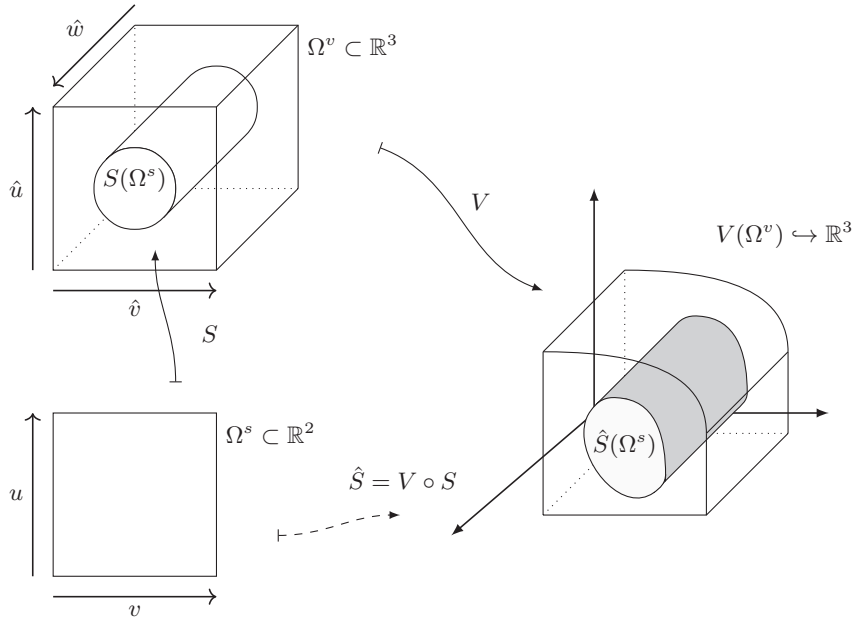


Figure 4.5: A sketch of parametric cylinder (2D) embedded in the parametric space of a volumetric Bézier cube (3D). The Bézier cube has been deformed.

4.3.2 Higher dimension objects

Furthermore, in the same way as the previous examples, we can define higher-order parametric objects, e.g., a parametric cylinder embedded in the parametric space of a volume. A differential sketch and rendering are visualized in Figures 4.5 and 4.6, respectively. Source code for the example is included in Appendix B.1.3. Primarily we utilize the functionality for prototyping and to inspect higher dimension differential geometry objects. However, these techniques could also be utilized to model higher dimension applications geometrically, such as object extraction from streams (4D), e.g., a single frame (2D) from a movie (4D), where *time* is the fourth dimension. This could for instance be utilized for stream-based up- or down-scaling.

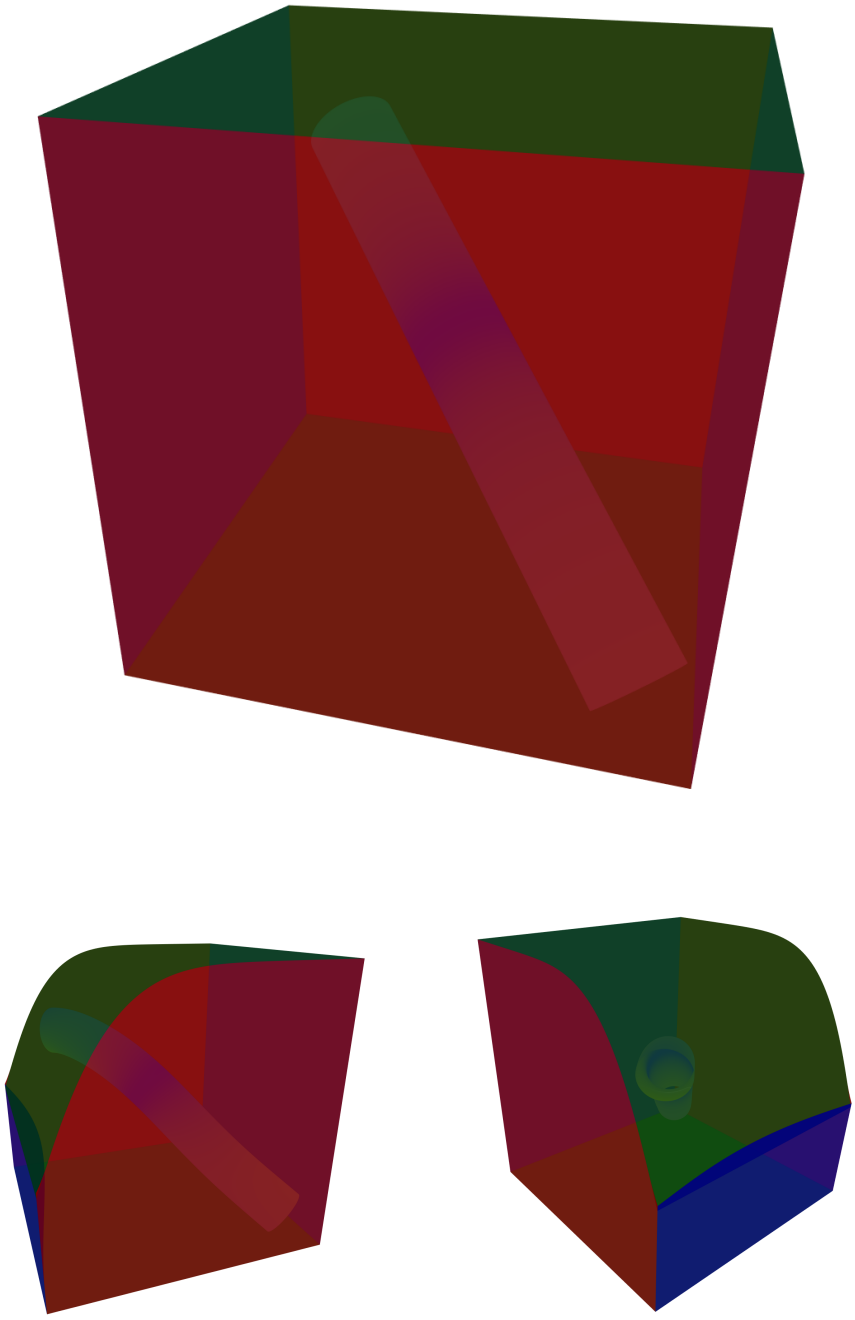


Figure 4.6: A parametric cylinder (2D) embedded in the parametric space of a volumetric Bézier cube (3D). The Bézier cube has been deformed.

4.3.3 Parametric polygon sub-object

As a final example, we have a sub-polygon. This is the recurring construction used in the blending spline polygon surface construction. The sub-polygon is illustrated in Figures 4.7 and 4.8, and a constructional source code example is included in Appendix B.1.4. An evaluation function for a sub-polygon surface can be defined as follows

```
1  static EvaluationResult
2  evaluate(const PSpacePolygonSurface&     pspace_polysurf,
3           const ParametricPolygonSurface* ppolysurf,
4           const typename PSpacePolygonSurface::PSpacePoint& par)
5  {
6    // Directional derivative step-size tolerance
7    constexpr auto step = 1e-5;
8
9    // function val
10   const auto pspace_eval = pspace_polysurf.evaluateParent(par, {0, 0});
11   const auto pobj_par
12     = blaze::subvector<0UL, ParametricPolygonSurface::PSpaceVectorDim>(
13       pspace_eval(0, 0));
14   const auto pobj_eval = ppolysurf->evaluateLocal(pobj_par, {0, 0});
15
16   // Data structure (1D)
17   EvaluationResult p(3);
18
19   // F
20   p(0) = pobj_eval(0);
21
22   // DuF
23   const auto Du = typename PSpacePolygonSurface::PSpaceVector{step, 0};
24   const auto [DuF_ppolysurf, DuF_ppolysurf_scale]
25     = D(pspace_polysurf, par, Du, *ppolysurf);
26   p(1) = DuF_ppolysurf * DuF_ppolysurf_scale;
27
28   // DvF
29   const auto Dv = typename PSpacePolygonSurface::PSpaceVector{0, step};
30   const auto [DvF_ppolysurf, DvF_ppolysurf_scale]
31     = D(pspace_polysurf, par, Dv, *ppolysurf);
32   p(2) = DvF_ppolysurf * DvF_ppolysurf_scale;
33
34   // Return data structure to higher level evaluator
35   return p;
36 }
```

The `pspace_polysurf` and `ppolysurf` input parameters are the polygon's sub-surface-type surface embedded in the parameter space of the original polygon surface and the original polygon surface, respectively. Furthermore, the `par` parameter provides the sub-object evaluator with information about the evaluation parameter position.

When evaluating the sub-surface, first, the parameter space polygon-surface is evaluated at the provided parameter position (line 10). Then the function value of the evaluation is interpreted as the parameter value of the original polygon surface, and this is evaluated (line 14). The evaluation result is then packed in the evaluation result data type and returned.

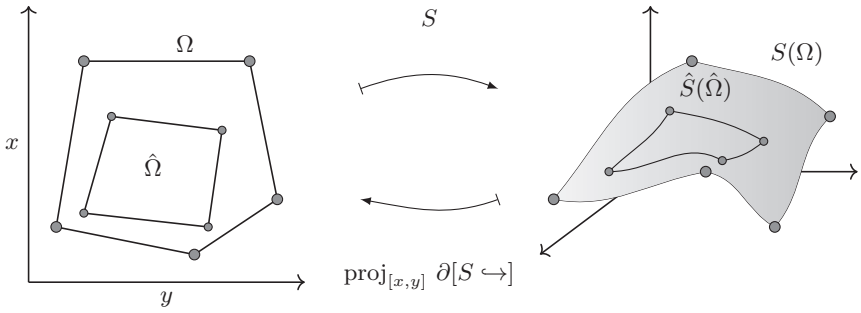


Figure 4.7: A quadrangular polygon embedded as a sub-polygon construction in the parametric domain of a pentagonal generalized Bézier patch with cubic ribbon-patches. $\partial[S \hookrightarrow]$ refers to the surface's set of embedded boundary coefficients.

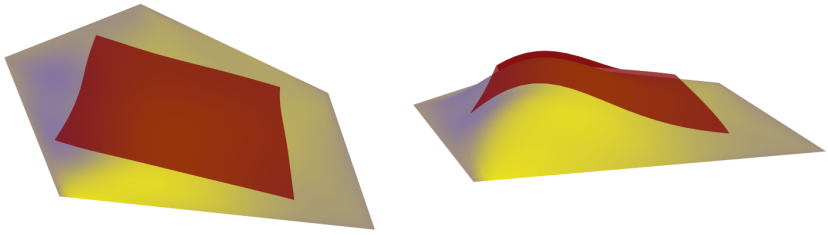


Figure 4.8: A quadrangular polygon embedded as a sub-polygon construction in the parametric domain of a pentagonal generalized Bézier patch with cubic ribbon-patches.

On lines (23–26) and (29–32), point-derivatives are computed along two primary axes of the underlying parametric coordinate system of the given space (2D). The differential operator, `D`, used on lines (25 and 31) is explained in the next section.

4.4 Compile-time enabled features

By having dimensions, space, and object meta-information available at compile-time, we can express various concepts statically. An example can be a derivation functor based on a forward finite difference operator

$$\nabla_h[f](x) = f(x+h) - f(x),$$

where we want to utilize the operator as follows

```
using ForwardDifference
  = FiniteDifference<kernel::ForwardDifferenceKernel<>>;

static constexpr ForwardDifference D{};

auto const f = [](const auto& p) {
  /* evaluate a function f at p */
};

Point  x = /* position  */;
Vector h = /* direction */;

auto const fDx = D(x, h, f);
```

We enable this functionality through a driver kernel and a difference-quotient function. The difference-quotient of a first degree finite differential operator can be implemented as follows

```
template <typename FDKernelOperator_T>
struct FiniteDifference {

  template <typename Point_T, typename Vector_T, typename Fn_T>
  auto operator()(Point_T&& t, Vector_T&& dt, Fn_T&& fn) const
  {
    const auto [a, b]
      = op(std::forward<Point_T>(t), std::forward<Vector_T>(dt),
           std::forward<Fn_T>(fn));
    return (a - b) / length(dt);
  }

private:
  static constexpr FDKernelOperator_T op{};
};
```

and the forward difference kernel as follows

```
namespace kernel
{
  template <typename = void>
  struct ForwardDifferenceKernel {

    template <typename Point_T, typename Vector_T, typename Fn_T>
    constexpr auto operator()(Point_T&& t, Vector_T&& dt, Fn_T fn) const
    {
      return fn(std::forward<Point_T>(t) + std::forward<Vector_T>(dt));
    }
  };
}
```

Using such techniques in combined with object-oriented compile-time programming, one can define expressions that produce smaller footprints and, provides added and better compile-time errors on failure. This is invaluable when prototyping complex constructions, such as parameterized polygonal surfaces, where for instance, partial derivatives are challenging to express explicitly. An example is provided in Appendix B.2, which shows how this is used to compute directional derivatives for sub-objects.

Chapter 5

Application examples in interactive geometric modeling

This chapter includes examples of blending spline applications, which help with highlighting some of the construction's main properties when applied to different areas of IGM. It builds a read thread connecting the papers included in the thesis, the peripheral work which inspired the research, and the work developed as a consequence. All peripheral work included in this section has either been published or communicated by the thesis author.

The following works are referenced in this chapter

Bratlie, J. and Dalmo, R. "Motion capture data represented using a blending type spline construction". In: *40th International conference applications of mathematics in engineering and economics AMEE14*. Ed. by Pasheva, V. and Venkov, G. Vol. 1631. AIP Conference Proceedings. AIP Publishing, 2014, pp. 153–157,

Bratlie, J., Dalmo, R., and Bang, B. *Evaluation of smooth spline blending surfaces using GPU*. Communication at the 8th International Conference on Curves and Surfaces, Paris, France. 2014,

Haavardsholm, B., Bratlie, J., and Dalmo, R. "Surface deformation over flexible joints using spline blending techniques". In: *ICNPAA 2014 World Congress: 10th International conference on Mathematical Problems in Engineering, Aerospace and Sciences*. Ed. by Sivasundaram, S. Vol. 1637. AIP Conference Proceedings. AIP Publishing, 2014, pp. 377–383,

Pedersen, A., Bratlie, J., and Dalmo, R. "Spline representation of connected surfaces with custom-shaped holes". In: *Large-Scale Scientific Computing 2015*. Ed. by Lirkov, I., Margenov, S., and Waśniewski, J. Vol. 9374. Lecture Notes in Computer Science. Springer, 2015, pp. 394–400,

Bratlie, J., Brustad, T. F., and Dalmo, R. *GPU-based rendering of blending spline surface lattices*. Communication at the 9th International Conference on Mathematical Methods for Curves and Surfaces, Tønsberg, Norway, June 23rd – 28th, 2016. 2016.

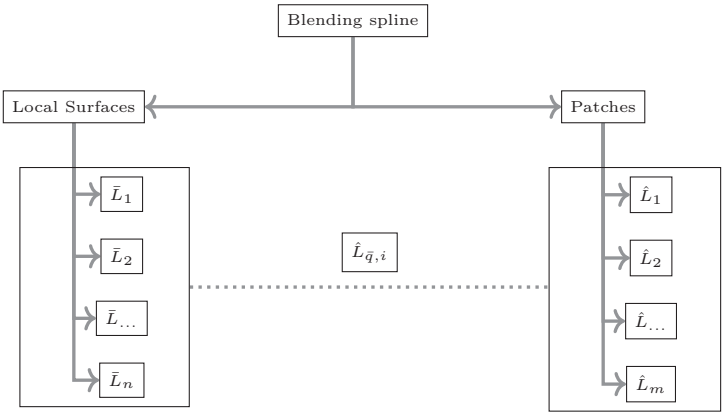


Figure 5.1: The spatial scenegraph for a blending spline construction, using notation from Paper V.

5.1 Spatial nature of blending spline constructions

The embedded blending spline construction can be visualized as a hierarchical scenegraph structure. Figure 5.1 utilizes a surface construction, as an example, to do just that. Imagine the blending spline surface itself as the root node of a scenegraph structure. As direct children, the surface will have two groups; local surfaces and blending spline patches. The blending spline patches are blending results from the local surfaces and are hierarchically considered global. The local surfaces exist one step lower in their own individual spaces. In our default implementations, these spaces are represented by homogeneous matrices. Furthermore, this constructional pattern can be repeated to construct multi-level constructions, where the local surfaces, themselves, are blending spline surfaces. Moreover, the local coefficients making up the data of the local surfaces will be the scenegraph's leaf nodes.

5.2 Categories of local geometry

The default focus point when discussing blending spline constructions is the type of B-function used, as this is one of the deciding smoothness factors. This, however, is discussed in Section 2.3.2.

Another deciding factor that influences both the smoothness of the construction and its editing capabilities is the choice of local geometry. In the following, we describe relevant categories of geometry, which is used as local geometry throughout the thesis, and group them by their typical features with respect to applications.

5.2.1 Free-form modeling

Local geometry used with free-form modeling is the usual suspects, such as Bézier and B-spline types. The main reason for this is their intuitive use of gadgets such as selectors and draggers.

5.2.2 Approximation

The local geometry constructions utilized for approximation are often chosen based on the problem at hand. If it is a polynomial problem, the choice is local polynomial constructions; if it is a trigonometric problem, one can choose trigonometric constructions, and so on. Other factors can be their similarities and characteristics with underlying data, expanding properties, or numerical stability. For instance, in Paper II, we utilize Taylor expanding polynomials, such that we can compare the overlying partitioning methods as the support in the local geometry expands. Another example is [HBD14], where we utilize Hermite surfaces to interpolate a required set of local coefficients.

5.2.3 Designed local geometry

By *designed local geometry*, we mean that we want to control the function value at a given knot. Due to the property of *vanishing derivatives*, the function and derivative values at a knot are equal to the evaluated values of the local geometry. An example is Hermite curves which, is used with keyframing of animation tracks to mirror the underlying data, as exemplified in Section 5.4.

5.2.4 Reuse local geometry coefficients

Another example is a technique touched upon in the same section, and which was one of the early ideas for Paper II, namely *reuse of local coefficients*. Given a data set, as the one making up the lower set of "curve coefficients" in Figure II.2, as an optimal configuration of coefficients, and imagine we utilize quadratic Bézier curves for the inner coefficient local curves and linear curves for the beginning and end local curves. Furthermore, imaging a coefficient distribution for local curves $\bar{C}_1, \ldots, \bar{C}_n$ as follows

c_1: p_1, p_2,

c_2: p_1, p_2, p_4,

c_3: p_2, p_4, p_7,

c_4: p_4, p_7, p_9,

c_5: p_7, p_9, p_{10},

c_6: p_9, p_{10}.

If we utilize a sufficiently smooth blending spline basis, the construction results in a C^1-continuous blending spline curve due to the continuity of the local curves.

Then, imagine an alternate example where the coefficients are optimized for local 2^{nd}- and 4^{th}-degree Hermite-type curves. The 2^{nd}-degree curves represent a start- and end-point in addition to a start- or end-tangent, dependent on whether it is used to represent a curve-coefficient at the start or end, respectively. The 4^{th}-degree curves represent a start-, middle- and end-point, and a start- and end-tangent. If the 2^{nd}-degree curves are used for the end-coefficient, and the 4^{th}-degree curves are used for the inner-coefficient local curves, we would have a coefficient distribution for local curves $\bar{C}_i, \ldots, \bar{C}_n$ as follows

c_1: $p_1, p_2, \vec{v}_2 = p_2 - p_1$,

c_2: $p_1, \vec{v}_1 = p_2 - p_1, p_2, p_4, \vec{v}_4 = p_4 - p_2$,

c_3: $p_2, \vec{v}_2 = p_4 - p_2, p_4, p_7, \vec{v}_7 = p_7 - p_4$,

c_4: $p_4, \vec{v}_4 = p_7 - p_4, p_7, p_9, \vec{v}_9 = p_9 - p_7$,

c_5: $p_7, \vec{v}_7 = p_9 - p_7, p_9, p_{10}, \vec{v}_{10} = p_{10} - p_7$,

c_6: $p_9, p_{10}, \vec{v}_9 = p_{10} - p_9$.

This results in a C^2-continuous blending spline curve due to the continuity of the local curves.

5.2.5 Reparameterization and type adoption

The last category we will discuss is local geometry constructions, used for reparameterization and type adoption. This category includes embedded sub-objects. In Paper V, we utilize this technique to blend adjacent sub-regions of local polygon constructions. Sub-objects can also be used to do type adoption as in [BBD16; Lak13], where specialized local constructions were adopted as tensor-product surfaces before blending. These techniques add a middle layer and are of a different geometric type than the global construction.

5.3 Blending spline surface skinning

In [HBD14], we described an experiment where blending spline surfaces were utilized for skinning. Skinning is the technique of wrapping and deforming a surface over a skeletal structure. The technique is closely coupled with a three-dimensional mesh or free form surface representing a virtual character in a

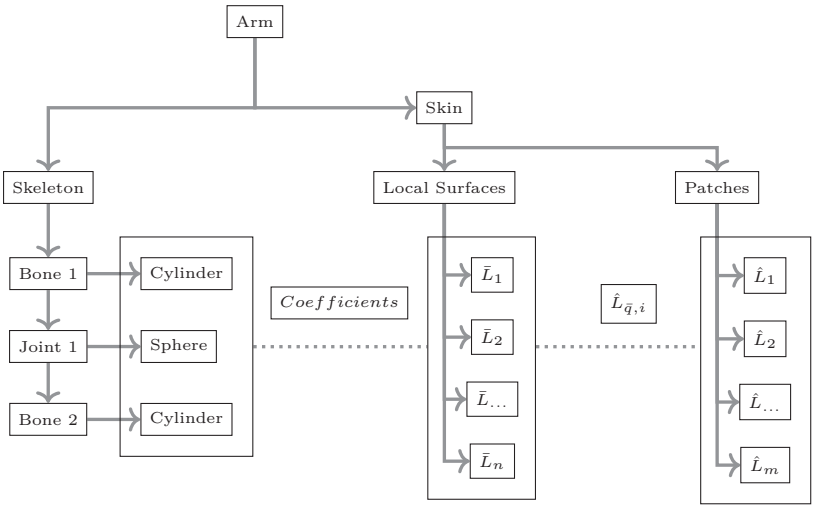


Figure 5.2: A principle spatial scenegraph for a blending spline surface skinning construction.

virtual world. This could be a person or an animal or an object such as a car or a house.

The principal experiment was an extracurricular sub-task to Haavardsholm's m.sc.-thesis [HBD14]. The author of this thesis proposed the fundamental idea, while the experiments and the published paper were in the majority written by Haavardsholm under supervision. Additionally, the paper was presented at a local conference, where it later was added to the proceedings and included as a part of Haavardsholm's m.sc.-thesis.

The prerequisites of the experiment were to see if the natural smoothing properties of the blending spline construction, combined with a free form deformation approach, no volume preservation, and custom local surfaces, would be suitable for skinning. The final skinning test case, included in the paper, was modeled over a fictitious minimal arm joint (two bones and a joint). The local geometry was built using 5^{th}-degree Hermite surfaces, which interpolate a start-, middle- and end-positional coefficient and a start-, middle- and end-1^{st}-degree derivative coefficient. One parameter line is mapped along the direction of the bones and joints, and one parameter line is mapped across. The individual Hermite surface coefficients were then embedded in the local spaces of the respective local arm parts. Furthermore, alignment and consistency are preserved as each logical arm part (bone or joint) represents its own independent space. The logical hierarchical organization is illustrated in Figure 5.2, and a selection of skinned surface visualizations are illustrated in Figure 5.3.

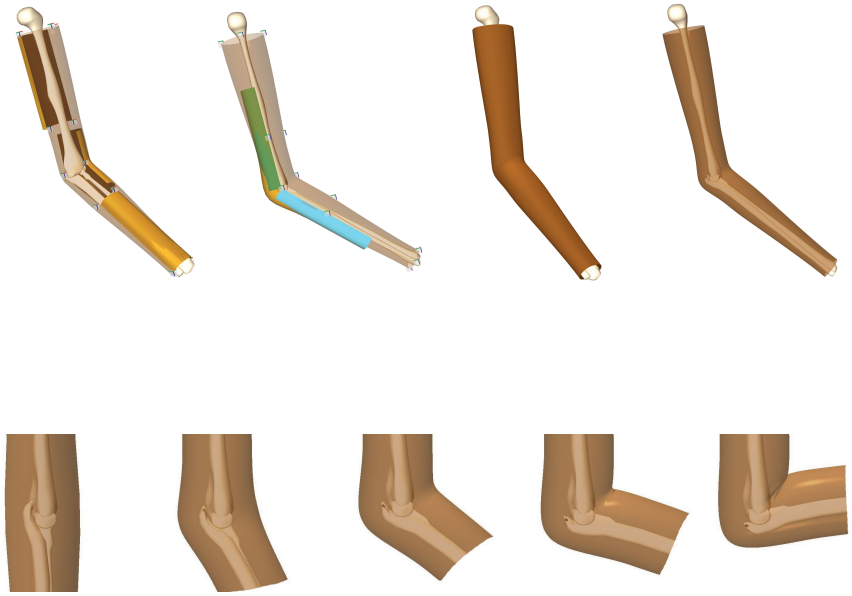


Figure 5.3: Visualization of a blending spline skinning surface over a fictitious minimal arm joint. Top left: a selection of local surfaces. Top right: The skinned surface from two offset angles; rendered opaque and translucent. Bottom: a time series visualizing the effect of the skinning method when deformed. Figures from [HBD14].

The results of the experiment demonstrate the natural effects of the shaping properties of blending splines. The skin self-intersects when it reaches a critical angle, seen from the latter figures in the bottom row time series in Figure 5.3. However, this is expected as there is no volume preservation. Moreover, since this is a free form deformation method, the method can be extended with external influencers and controls such as positional control from muscles, organs, or volume preservation controls, such as simulating character animation. This can be applied to both the local geometry coefficients or to the space of the local geometry itself.

5.4 Utilization to animation tracks

Keyframing is the process of interpolating key scene state positions, with respect to time along an animation track. Examples could, for instance, be when two cars collide during an intermediate sequence in a semi-interactive car game cutscene, or when a sword hits a shield in a sword-fight scene, or when the foot of a humanoid model hits the floor when walking in a cyclic walking animation. The keyframing process is achieved by synchronizing the animation tracks at

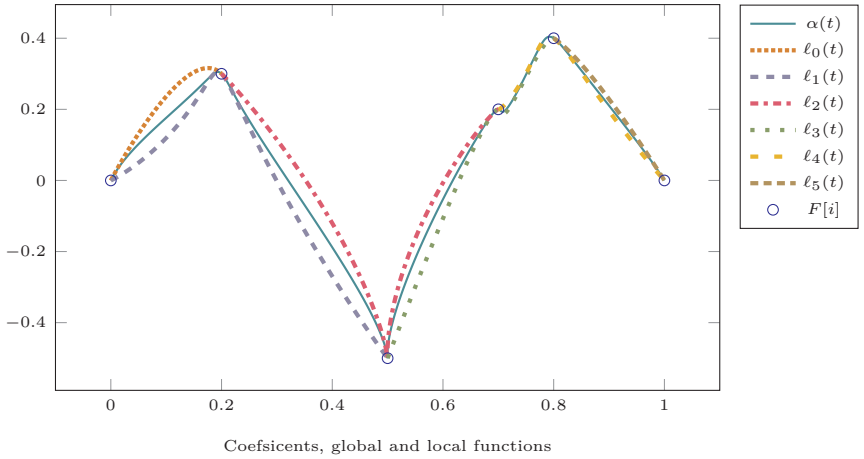


Figure 5.4: Blending spline animation track utilizing the LERB basis and local 3^{rd}- and 5^{th}-degree Hermite curves for boundary and inner local functions, respectively. The animation track uses the technique where the local functions reuses the local coefficients. The first, third, and last local functions have zero speed.

specific global time points while the intermediate positions are interpolated.

Keyframing is an essential part of skin (Section 5.3) animation. Another is rigging, which is the process of associating the movable coefficients of skinning with the different bones of a skeletal animation model. The skinning application example from Section 5.3 solves this by embedding the local surface coefficients in respective bone spaces. When animating, each bone in the model is moved along respective time-parameterized curves (animation tracks). The skinning coefficients are then updated through the rig.

The de facto industry standard used for keyframe interpolation is a trade-off between Catmull-Rom spline interpolation (cubic B-spline) [CR74] and spherical linear interpolation (Slerp) [Sho85], where the Slerp method is used if the time-step between two animation frames are sufficiently small.

In [BD14], we investigated blending splines curves and its potential for representation of animation tracks. In addition we applied a local curve method where we reused the local coefficients when building local curves, as first investigated in Paper II [BDZ14] as the inflexion method. In Figure 5.4, we have plotted an example positional two-dimensional track (embed space) together with the track's local curves. The construction uses a different type of local Hermite type curves to represent the underlying dataset, which is then blended by the blending spline curve construction. This enables us to control, for instance, the approach and speed at each knot.

In the top part of Figure 5.5 we plot the curve together with the first

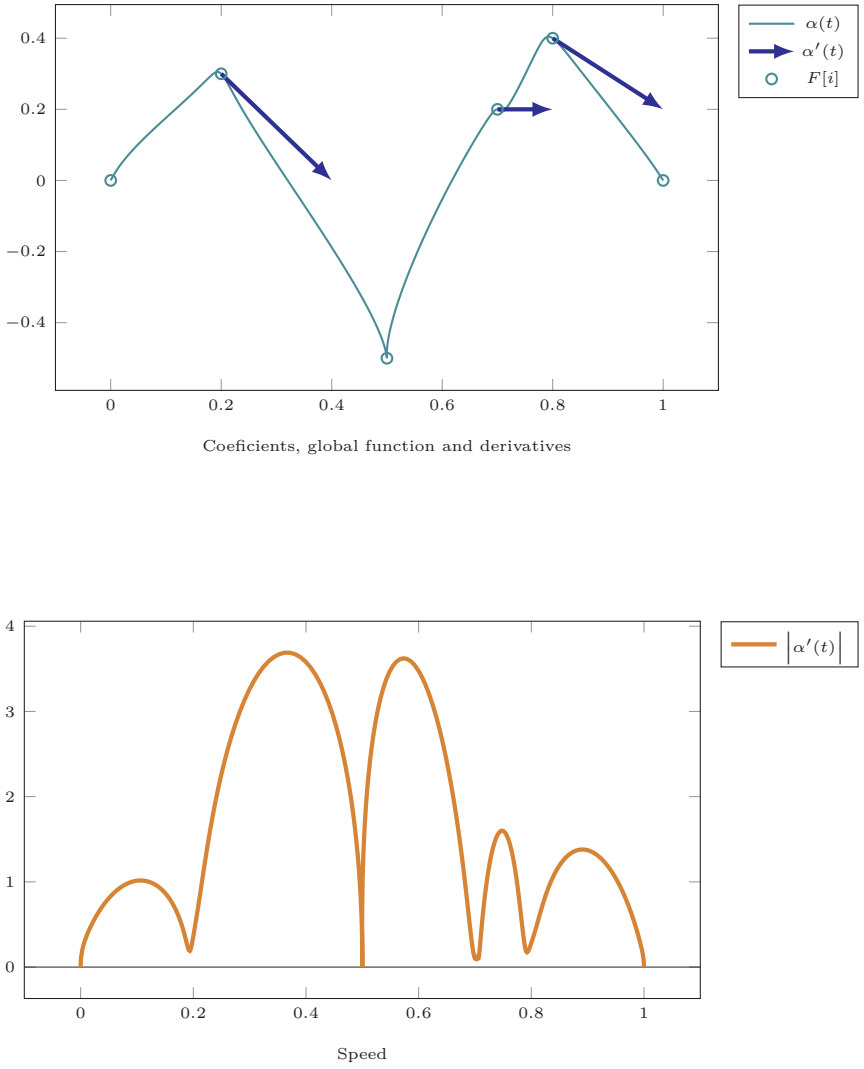


Figure 5.5: Derivative and speed plots of the animation track from Figure 5.4.

derivatives at each knot, which are interpolated by the local curves. The bottom part of Figure 5.5 plots the magnitude of the first derivative along the animation track curve. We clearly see here how the continuous blending spline curve has zero speed at the first, third and last knot.

From an *animation point of view*, we can make the animation stop and turn while controlling the approach. Moreover, this means we can construct and control natural motion in animations, e.g., a sprinter starts to run at zero speed

and naturally speeds up. This information can then also be evaluated along the animation curve.

5.4.1 Future work and applications

By utilizing the blending spline construction properties, we can employ the construction with animation applications, which is an interesting topic for future research

The spatial and hierarchical nature of the blending spline construction, which introduces enriched local constructions, in this case, curves, means that the local coefficients are embedded in a space having more than positional properties, such as orientation. This gives us the possibility to not only apply spatial transformations to the local coefficients as interactive extensions, but we can represent the whole animation track composed with different features. For instance, we could build in minimal rotation as part of the local curves or the global curve construction.

Another application is, as briefly mentioned earlier, we can control the keyframing closer by providing more control to the local "curves" in the form of derivatives, which lets us model not just the interpolation of positional keyframes but also properties such as velocity, acceleration, and more. If applied to science-fiction, the curve can also warp and still be mathematically smooth by introducing double knots. This is briefly touched upon in Section 5.6, however, applied to surfaces.

Applying the construction and utilizing sub-curves as local curves, we can make animated objects follow paths in other embeddable geometric objects. For instance

- in a simulation where a character is walking, its foot can perfectly align with a surface, or
- an animated car can controllably follow a path interpreted as a sub-curve embedded in a surface, or
- an object can follow a path in a vector field simulation.

As a tangent into a different application field; machining. A robot constructing an object by utilizing a given tool on some material, such as wood or metal, needs to design a machining path based on the given tool, material, and wear and tear. If these properties are provided to a blending spline curve, the machining track (animation track) could customize the local curve such that it controls not just the position of the tools approach but also its spatial orientation.

5.5 Iso-geometric modeling and visualization

Paper III was the first paper that interpreted the blending spline construction as a patchwork, see Section 2.1, however implicitly. The paper introduced a patch-wise GPU-centric rendering technique for blending spline surfaces and exemplified

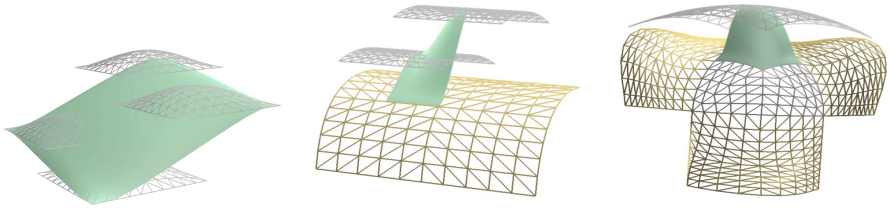


Figure 5.6: Blending spline surfaces over principle test-lattice configurations. The blending spline patch is shaded in a solid shade of green, and the local surfaces are rendered as wireframe surfaces. Left: patch with regular loci only. Middle: patch with regular and t-type loci. Right: patch with regular and star-type loci.

the rendering technique by relating it to irregular tensor-product constructions. However, the prototype at this stage was developed for regular tensor-product grids. Following, in communication at "The 9th International Conference on Mathematical Methods for Curves and Surfaces" [BBD16], an updated prototype was presented. Here the rendering technique from Paper III was fully applied to the irregular-grid tensor-product surface construction introduced in [Lak13].

The construction example in [Lak13] trades modeling-stiffness for the possibility to represent a minimal support blending spline tensor-product surface over irregular grids. An irregular tensor-product spline grid contains irregular knot joints (this is where tensor-product parameter lines join irregularly). Irregular knots are categorized into three different knot types; regular, t- and star-joints (see [Lak13]).

The local surface cover at irregular knots were expanded by custom local surface constructions. Then the local surfaces used in the blending spline surface were represented in the form of tensor-product sub-surfaces. These sub-surface were embedded in and evaluated through the parameter space of the custom local surface constructions. Figure 5.6 shows rendered illustrations of blending spline surfaces created over the principal test-lattices, as described in [Lak13] and Figure III.3.

These works were the inspiration and first steps towards Paper V.

5.5.1 GPU implementation

The paper included in Paper III focused on two primary aspects; the conceptual partitioning of the irregular grids, described in [Lak13], and its connection to the blending spline patch and GPU-centric rendering method. The accompanying prototype used as an internal proof of concept was used to generate the illustrations for the poster session at [BDB14] and the resulting paper.

The illustrations in Figures 5.6 and 5.7 are produced with the full prototype constructed for [BBD16]. The prototype was implemented in C++ and GLSL

as a rendering layer on top of GMlib [LBK06]. The geometric structure of the construction was handled on the host side, where OpenGL [KHR2-] rendering pipelines were constructed by building specialized tessellation evaluation shader kernels. Each kernel is tailored for a given configuration of local surfaces. This means type, spline degree, and other non-runtime properties. For instance, the three quad-configurations in Figure 5.6 yields:

Regular loci (left)
: local surfaces are of quadratic Bézier-type and are each covering the local neighborhood of its associated regular loci.

T-joint (middle)
: Two of the local surfaces are of quadratic Bézier-type, and are each covering the local neighborhood of its associated regular loci. The two other local surfaces are of different local sub-surface types. The local sub-surfaces are embedded in the parametric space of a common quadratic Bézier-type surface. This surface construction is covering the associated t-type locus neighborhood.

Star-joint (right)
: One of the local surfaces is of quadratic Bézier-type, and is covering the local neighborhood of its associated regular locus. The three remaining local surface patches are of different local sub-surface types. The local sub-surfaces are embedded in a common Bézier-type designed patchwork. This patchwork is then covering its associated star-type locus neighborhood.

5.5.2 Conceptual simulation cost

The principal cost of the GPU-implementation is constant after construction. This means that the surface patch tessellation task is allocated to and executed by the GPU, and is therefore freed from the CPU. Only the local surface modelview matrices and local surface spline-coefficient data are pushed from the CPU-host side to the GPU-client side, with respect to animated blending splines. As this data footprint is small and constant, it will lead to a constant, predictable, and low bandwidth cost in the CPU-to-GPU transport layer. Moreover, it frees up bandwidth for other resources and processes such as physical simulations, shading, and other effects.

The illustration in Figure 5.7 is a simulation snapshot of a blending spline surface consisting of 400 local quadratic Bézier-type surfaces. Every single coefficient of each local surface is changed in a "random" trigonometric pattern, and every individual local surface is transformed by an affine transformation; every frame. The example was created for and demonstrated at [BBD16].

5.6 Designing surface boundaries using warping

By combining two basis function properties, *vanishing derivatives* and *minimal support*, with double knots and designed local geometry we can engineer

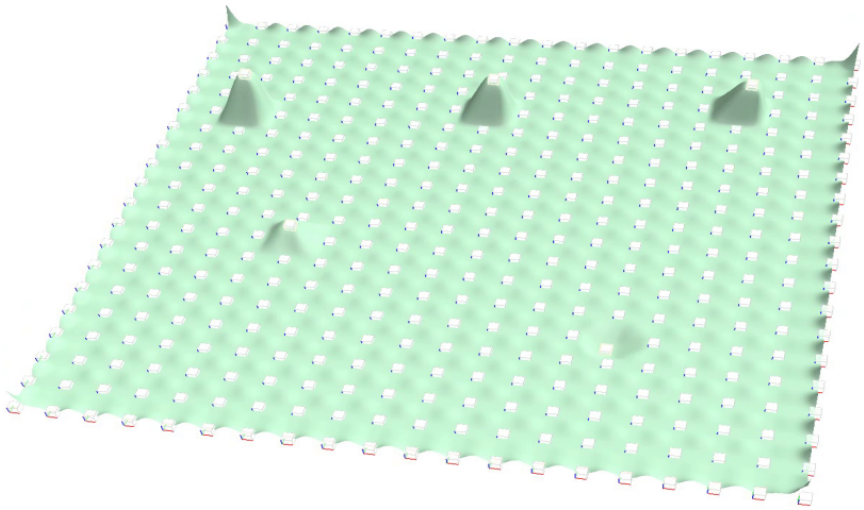


Figure 5.7: A simulation snapshot of a blending spline surface consisting of 400 (20×20) local quadratic Bézier surfaces. Every single coefficient of each local surface is changed in a "random" trigonometric pattern, and every individual local surface is transformed by an affine transformation; every frame. This local simulation was locally nicknamed the "dancing surface".

constructions with warps along a given parameter line. This results in constructions which are geometrically discontinuous, G^0-smooth (e.g., contains holes and cavities), but which at the same time are mathematically continuous, C^k-smooth, where k is decided as a minimum over the smoothness of the blending spline basis and local geometry constructions.

In [PBD15], an application construction was proposed which utilizes free-form curves, as input to local Coons' patches, to design smooth surfaces with holes. The principal experiment was an extracurricular sub-task to Pedersen's m.sc.-thesis [Ped14], where it was proposed as a possible construction for use with terminal ballistics and IGA [HCB05]. The author of this thesis proposed the fundamental idea, while the experiments and the paper were in the majority written and developed into a conference paper,, [PBD15], by Pedersen, under supervision. The paper proposes to build local quadratic Coons' patches from a combination of inner and outer boundary curves, where the inner boundary curves are of free-form type. Furthermore, the construction utilizes an ERBS basis. This results in a C^1-smooth surface construction. Elements of the construction are visualized in Figure 5.8.

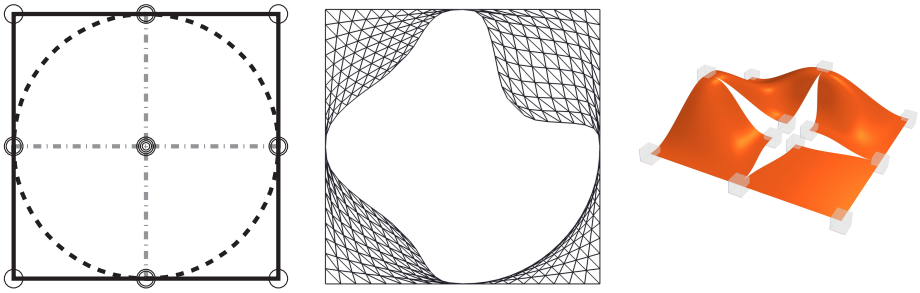


Figure 5.8: Tensor-product blending spline surface construction with an internal hole. Left-to-right; A schematic illustration of a tensor-product patch domain with inscribed potential inner boundary (dotted), double-knot knot vector lines (dash-dotted), and number of associated double-knot local geometry surfaces (rings). A wireframe rendering of a tensor-product surface with a free-form-shaped hole. A rendered tensor-product surface where the four local surfaces, all associated with the central knot, have been translated towards the center of their respective patch. Figures from [PBD15].

Bibliography

[And08] Andresen, K. M. “Brukergrensesnitt for kunstnerisk design ved bruk av ERBS”. MA thesis. Narvik University College, 2008 (cit. on p. 2).

[Bot+02] Botsch, M., Steinberg, S., Bischoff, S., and Kobbelt, L. “OpenMesh: A Generic and Efficient Polygon Mesh Data Structure”. In: *OpenSG Symposium 2002*. 2002 (cit. on p. 34).

[Bra11] Bratlie, J. “Methods for userguided compression algorithms”. In: *NIK: Norsk Informatikkonferanse*. Vol. 2011. 2011 (cit. on pp. 1, 10).

[Bra13] Bratlie, J. “Local refinement of GERBS surfaces with applications to interactive geometric modeling”. In: *39th International conference applications of mathematics in engineering and economics AMEE13*. Ed. by Pasheva, V. and Venkov, G. Vol. 1570. AIP Conference Proceedings. AIP Publishing, 2013, pp. 18–25 (cit. on p. 5).

[BBD16] Bratlie, J., Brustad, T. F., and Dalmo, R. *GPU-based rendering of blending spline surface lattices*. Communication at the 9$^{\text{th}}$ International Conference on Mathematical Methods for Curves and Surfaces, Tønsberg, Norway, June 23$^{\text{rd}}$ – 28$^{\text{th}}$, 2016. 2016 (cit. on pp. 2, 4, 47, 50, 56, 57).

[BD14] Bratlie, J. and Dalmo, R. “Motion capture data represented using a blending type spline construction”. In: *40th International conference applications of mathematics in engineering and economics AMEE14*. Ed. by Pasheva, V. and Venkov, G. Vol. 1631. AIP Conference Proceedings. AIP Publishing, 2014, pp. 153–157 (cit. on pp. 2, 5, 10, 47, 53).

[BD19] Bratlie, J. and Dalmo, R. “Exploring future C++ features within a geometric modeling context”. In: *NIK: Norsk Informatikkonferanse*. Vol. 2019. 2019 (cit. on p. 6).

[BD21] Bratlie, J. and Dalmo, R. “Blending spline polygon surface over arbitrary poly-mesh topology”. Under revision. (Cit. on pp. 6, 33).

[BDB14] Bratlie, J., Dalmo, R., and Bang, B. *Evaluation of smooth spline blending surfaces using GPU*. Communication at the 8$^{\text{th}}$ International Conference on Curves and Surfaces, Paris, France. 2014 (cit. on pp. 47, 56).

[BDB15a] Bratlie, J., Dalmo, R., and Bang, B. “Evaluation of smooth spline blending surfaces using GPU”. In: *Curves and surfaces. 8th International Conference*. Ed. by Boissonnat, J.-D., Cohen, A., Gibaru, O., Gout, C., Lyche, T., Mazure, M.-L., and Schumaker, L. L. Vol. 9213. Lecture Notes in Computer Science. Springer, 2015, pp. 60–69 (cit. on pp. 5, 33).

[BDB15b] Bratlie, J., Dalmo, R., and Bang, B. “Wavelet compression of spline coefficients”. In: *Numerical Methods and Applications 2014*. Ed. by Dimov, I., Fidanova, S., and Lirkov, I. Vol. 8962. Lecture Notes in Computer Science. Springer, 2015, pp. 246–253 (cit. on p. 11).

[BDZ14] Bratlie, J., Dalmo, R., and Zanaty, P. “Fitting of Discrete Data with GERBS”. In: *Large-Scale Scientific Computing 2013*. Ed. by Lirkov, I., Margenov, S., and Waśniewski, J. Vol. 8353. Lecture Notes in Computer Science. Springer, 2014, pp. 569–576 (cit. on pp. 5, 53).

[Bud+16] Budninskiy, M., Liu, B., Tong, Y., and Desbrun, M. “Power Coordinates: A Geometric Construction of Barycentrc Coordinates on Convex Polytopes”. In: *ACM Transactions on Graphics* vol. 35, no. 6 (2016), p. 11 (cit. on p. 9).

[CR74] Catmull, E. and Rom, R. “A class of local interpolating splines”. In: *Computer Aided Geometric Design*. Ed. by Barnhill, R. and Riesenfeld, R. Academic Press, New York, 1974, pp. 317–326 (cit. on p. 53).

[Coo67] Coons, S. A. *Surfaces for Computer-Aided Design of Space Forms*. Project MAC-TR-41. Massachusetts, USA: Massachusetts Institute of Technology, 1967 (cit. on p. 13).

[CLM05] Costantini, P., Lyche, T., and Manni, C. “On a class of weak Tchebycheff systems”. In: *Numerische Mathematik* vol. 101, no. 2 (2005). cited By 80, pp. 333–354 (cit. on p. 20).

[DB14a] Dalmo, R. and Bratlie, J. “Data approximation using a blending type spline construction”. In: *40th International conference applications of mathematics in engineering and economics AMEE14*. Ed. by Pasheva, V. and Venkov, G. Vol. 1631. AIP Conference Proceedings. AIP Publishing, 2014, pp. 147–152 (cit. on p. 10).

[DB14b] Dalmo, R. and Bratlie, J. “Discrete Wavelet Compression of ERBS”. In: *Large-Scale Scientific Computing 2013*. Ed. by Lirkov, I., Margenov, S., and Waśniewski, J. Vol. 8353. Lecture Notes in Computer Science. Springer, 2014, pp. 577–584 (cit. on p. 10).

[DBB15] Dalmo, R., Bratlie, J., and Bang, B. “Performance of a wavelet shrinking method”. In: *Numerical Methods and Applications 2014*. Ed. by Dimov, I., Fidanova, S., and Lirkov, I. Vol. 8962. Lecture Notes in Computer Science. Springer, 2015, pp. 262–270 (cit. on p. 11).

[Dal+14] Dalmo, R., Bratlie, J., Bang, B., and Lakså, A. "Smooth spline blending surface approximation over a triangulated irregular network". In: *International Journal of Applied Mathematics* vol. 27, no. 1 (2014), pp. 109–119 (cit. on p. 10).

[DBZ14] Dalmo, R., Bratlie, J., and Zanaty, P. "Image processing with LERBS". In: *ICNPAA 2014 World Congress: 10th International conference on Mathematical Problems in Engineering, Aerospace and Sciences*. Ed. by Sivasundaram, S. Vol. 1637. AIP Conference Proceedings. AIP Publishing, 2014, pp. 271–278 (cit. on p. 11).

[DBZ15] Dalmo, R., Bratlie, J., and Zanaty, P. "Image compression using an adjustable basis function". In: *Mathematics in Engineering, Science and Aerospace* vol. 6, no. 1 (2015), pp. 25–34 (cit. on p. 11).

[de 62] de Boor, C. "Bicubic Spline Interpolation". In: *Journal of Mathematics and Physics* vol. 41, no. 1-4 (1962), pp. 212–218 (cit. on p. 13).

[DBL09] Dechevsky, L. T., Bang, B., and Lakså, A. "Generalized Expo-Rational B-Splines". In: *International Journal of Pure and Applied Mathematics* vol. 57, no. 6 (2009), pp. 833–872 (cit. on p. 14).

[DBG12] Dechevsky, L. T., Bratlie, J., and Gundersen, J. "Index Mapping between Tensor-Product Wavelet Bases of Different Number of Variables, and Computing Multivariate Orthogonal Discrete Wavelet Transforms on Graphics Processing Units". In: *Large-Scale Scientific Computing 2011*. Ed. by Lirkov, I., Margenov, S., and Waśniewski, J. Vol. 7116. Lecture Notes in Computer Science. Springer, 2012, pp. 402–410 (cit. on p. 10).

[Dec10] Dechevsky, L. T. "Beta-function B-splines: definition and basic properties". In: *International Journal of Pure and Applied Mathematics* vol. 65, no. 3 (2010), pp. 279–295 (cit. on pp. 14, 19).

[Dec+11] Dechevsky, L. T., Bratlie, J., Bang, B., Lakså, A., and Gundersen, J. "Wavelet-based lossless one- and two-dimensional representation of multidimensional geometric data". In: *37th International conference applications of mathematics in engineering and economics AMEE11*. Ed. by Venkov, G., Kovacheva, R., and Pasheva, V. Vol. 1410. AIP Conference Proceedings 1. AIP Publishing, 2011, pp. 83–97 (cit. on p. 10).

[DLB04] Dechevsky, L. T., Lakså, A., and Bang, B. "NUERBS form of expo-rational B-splines". In: Preprint vol. 1 (2004). ISSN 1504-4653, p. 18 (cit. on p. 14).

[DLB06] Dechevsky, L. T., Lakså, A., and Bang, B. "Expo-Rational B-Splines". In: *International Journal of Pure and Applied Mathematics* vol. 27, no. 3 (2006), pp. 319–362 (cit. on pp. 13, 14, 17).

[DZ13] Dechevsky, L. T. and Zanaty, P. "Smooth GERBS, orthogonal systems and energy minimization". In: *39th International conference applications of mathematics in engineering and economics AMEE13*. Ed. by Pasheva, V. and Venkov, G. Vol. 1570. AIP Conference Proceedings. AIP Publishing, 2013, pp. 135–162 (cit. on p. 17).

[Fer64] Ferguson, J. "Multivariable Curve Interpolation". In: *Journal of the Association for Computing Machinery* vol. 11, no. 2 (1964), pp. 221–228 (cit. on p. 13).

[Gor69a] Gordon, W. "Blending-function methods of bivariate and multivariate interpolation and approximation". In: *SIAM Journal on Numerical Analysis* vol. 8 (1969), pp. 158–177 (cit. on p. 13).

[Gor69b] Gordon, W. "Spline-blended surface interpolation through curve networks". In: *J of Math. and Mechanics* vol. 18 (1969), pp. 931–952 (cit. on p. 13).

[Gre74] Gregory, J. A. "Smooth Interpolation Without Twist Constraints". In: *Computer Aided Geometric Design*. Ed. by Barnhill, R. E. and Riesenfeld, R. F. Academic Press, 1974, pp. 71–87 (cit. on p. 13).

[HBD14] Haavardsholm, B., Bratlie, J., and Dalmo, R. "Surface deformation over flexible joints using spline blending techniques". In: *ICNPAA 2014 World Congress: 10th International conference on Mathematical Problems in Engineering, Aerospace and Sciences*. Ed. by Sivasundaram, S. Vol. 1637. AIP Conference Proceedings. AIP Publishing, 2014, pp. 377–383 (cit. on pp. 2, 11, 47, 49–52).

[Har01] Hartmann, E. "Parametric G^n blending of curves and surfaces". In: *The Visual Computer* vol. 17, no. 1 (Feb. 2001), pp. 1–13 (cit. on pp. 13, 19).

[HK18] Hettinga, G. J. and Kosinka, J. "Multisided generalisations of Gregory patches". In: *Computer Aided Geometric Design* vol. 62 (2018), pp. 166–180 (cit. on p. 13).

[Hu+19a] Hu, Y., Anderson, L., Li, T.-M., Sun, Q., Carr, N., Ragan-Kelley, J., and Durand, F. "DiffTaichi: Differentiable Programming for Physical Simulation". In: *arXiv preprint arXiv:1910.00935* (2019) (cit. on p. 8).

[Hu+19b] Hu, Y., Li, T.-M., Anderson, L., Ragan-Kelley, J., and Durand, F. "Taichi: A Language for High-Performance Computation on Spatially Sparse Data Structures". In: *ACM Transactions on Graphics* vol. 38, no. 6 (Nov. 2019) (cit. on p. 8).

[HCB05] Hughes, T. J. R., Cottrell, J. A., and Bazilevs, Y. "Isogeometric Analysis: CAD, Finite Elements, NURBS, Exact Geometry and Mesh Refinement". In: *Computer Methods in Applied Mechanics and Engineering* vol. 194, no. 39-41 (Oct. 2005), pp. 4135–4195 (cit. on p. 58).

[Igl+12] Iglberger, K., Hager, G., Treibig, J., and Rüde, U. "Expression Templates Revisited: A Performance Analysis of Current Methodologies". In: *SIAM Journal on Scientific Computing* vol. 34, no. 2 (2012), pp. C42–C69 (cit. on p. 34).

[Jac13] Jacobson, A. "Bijective Mappings with Generalized Barycentric Coordinates: A Counterexample". In: *Journal of Graphics Tools* vol. 17, no. 1-2 (2013), pp. 1–4 (cit. on p. 26).

[KHR2-] KHRONOS group. *OpenGL*. 1992-. URL: http://khronos.org/opengl (cit. on pp. 33, 57).

[Kra19] Kravetc, T. "Finite element method application of ERBS triangles". In: *NIK: Norsk Informatikkonferanse*. Vol. 2019. 2019 (cit. on pp. 14, 23).

[Kra20] Kravetc, T. "Representation and application of spline-based finite elements". PhD thesis. UiT The Arctic University of Norway, 2020 (cit. on pp. 14, 23).

[Lak07] Lakså, A. "Basic properties of Expo-Rational B-splines and practical use in Computer Aided Geometric Design". (Dr.philos.) PhD thesis. University of Oslo, 2007 (cit. on pp. 2, 3, 9, 14, 17, 25).

[Lak13] Lakså, A. "ERBS-surface construction on irregular grids". In: *39th International conference applications of mathematics in engineering and economics AMEE13*. Ed. by Pasheva, V. and Venkov, G. Vol. 1570. AIP Conference Proceedings. AIP Publishing, 2013, pp. 113–120 (cit. on pp. 3, 5, 14, 50, 56).

[Lak14] Lakså, A. "Construction and properties of non-polynomial spline curves". In: *ICNPAA 2014 World Congress: 10th International conference on Mathematical Problems in Engineering, Aerospace and Sciences*. Ed. by Sivasundaram, S. Vol. 1637. AIP Conference Proceedings. AIP Publishing, 2014, pp. 545–554 (cit. on pp. 3, 14, 20, 21).

[LB15] Lakså, A. and Bang, B. "Surface construction on irregular grids". In: *Large-Scale Scientific Computing 2015*. Ed. by Lirkov, I., Margenov, S., and Waśniewski, J. Vol. 9374. Lecture Notes in Computer Science. Springer, 2015, pp. 385–393 (cit. on p. 14).

[LBK06] Lakså, A., Bang, B., and Kristoffersen, A. R. *GM_lib, a C++ library for geometric modeling*. Narvik, Norway: Narvik University College, 2006 (cit. on pp. 33, 57).

[LD89] Loop, C. T. and DeRose, T. D. "A Multisided Generalization of Bézier Surfaces". In: *ACM Transactions on Graphics* vol. 8, no. 3 (1989), pp. 204–234 (cit. on p. 13).

[MC17] Majeed, M. and Cirak, F. “Isogeometric analysis using manifold-based smooth basis functions”. In: *Computer Methods in Applied Mechanics and Engineering* vol. 316 (2017). Special Issue on Isogeometric Analysis: Progress and Challenges, pp. 547–567 (cit. on p. 14).

[MPS11] Manni, C., Pelosi, F., and Sampoli, M. L. “Generalized B-splines as a tool in isogeometric analysis”. In: *Computer Methods in Applied Mechanics and Engineering* vol. 200, no. 5 (2011), pp. 867–881 (cit. on p. 20).

[NG00] Navau, J. and Garcia, N. “Modelling surfaces from planar irregular meshes”. In: *Computer Aided Geometric Design* vol. 17, no. 1 (2000), pp. 1–15 (cit. on p. 19).

[Olo19] Olofsen, H. “Blending functions based on trigonometric and polynomial approximations of the Fabius function”. In: *NIK: Norsk Informatikkonferanse*. Vol. 2019. 2019 (cit. on pp. 17, 19, 23).

[Ped14] Pedersen, A. “Shooting simulator”. Faculty of Technology. MA thesis. Narvik University College, 2014 (cit. on p. 58).

[PBD15] Pedersen, A., Bratlie, J., and Dalmo, R. “Spline representation of connected surfaces with custom-shaped holes”. In: *Large-Scale Scientific Computing 2015*. Ed. by Lirkov, I., Margenov, S., and Waśniewski, J. Vol. 9374. Lecture Notes in Computer Science. Springer, 2015, pp. 394–400 (cit. on pp. 11, 47, 58, 59).

[Rah+21] Rahtz, S. et al. *The TEX Live Guide*. May 2021. URL: http://tug.org/texlive/ (cit. on p. 7).

[Ras06] Rasmussen, E. “Improved visulization of scalar data using interpolation”. MA thesis. Narvik University College, 2006 (cit. on pp. 2, 23).

[SV18] Salvi, P. and Varady, T. “Multi-sided Bezier surfaces over concave polygonal domains”. In: *Computers & Graphics* vol. 74 (2018), pp. 56–65 (cit. on pp. 13, 25, 26).

[SHF13] Schneider, T., Hormann, K., and Floater, M. S. “Bijective Composite Mean Value Mappings”. In: *Computer Graphics Forum* vol. 32, no. 5 (2013), pp. 137–146 (cit. on p. 26).

[Sch17] Schneider, T. “Theory and Applications of Bijective Barycentric Mappings”. PhD thesis. Universita della Svizzera, 2017 (cit. on p. 26).

[Sed+03] Sederberg, T. W., Zheng, J., Bakenov, A., and Nasri, A. “T-splines and T-NURCCs”. In: *ACM SIGGRAPH 2003 Papers*. SIGGRAPH ’03. San Diego, California: ACM, 2003, pp. 477–484 (cit. on p. 3).

[SLY05] Séquin, C. H., Lee, K., and Yen, J. “Fair, G^2- and C^2-continuous circle splines for the interpolation of sparse data points”. In: *Computer Aided Design* vol. 37, no. 2 (2005), pp. 201–211 (cit. on p. 13).

[Sho85] Shoemake, K. “Animating Rotation with Quaternion Curves”. In: *ACM SIGGRAPH 1985 Papers*. Vol. 19. SIGGRAPH ’85 3. San Francisco, California: ACM, 1985, pp. 245–254 (cit. on p. 53).

[The19] The Qt Company. *Qt - Complete software development framework*. 2019. URL: http://qt.io (cit. on p. 33).

[TZ11] Tosun, E. and Zorin, D. “Manifold-based surfaces with boundaries”. In: *Computer Aided Geometric Design* vol. 28, no. 1 (2011), pp. 1–22 (cit. on p. 14).

[UiT21a] UiT - The Archtic University of Norway. *Simulations*. R&D group in mathematical and geometrical modeling, numerical simulations, programming and visualization. 2021. URL: http://en.uit.no/forskning/forskningsgrupper/gruppe?p_document_id=471206 (cit. on p. 1).

[UiT21b] UiT - The Arctic University of Norway. *GMlib/GMlib2 C++ Geometric modeling library*. (NeIC). May 2021. URL: http://source.coderefinery.org/gmlib (cit. on p. 33).

[VRS11] Várady, T., Rockwood, A., and Salvi, P. “Transfinite surface interpolation over irregular n-sided domains”. In: *Computer Aided Design* vol. 43, no. 11 (2011). Solid and Physical Modeling 2011, pp. 1330–1340 (cit. on p. 26).

[VSK16] Várady, T., Salvi, P., and Karikó, G. “A Multi-sided Bézier Patch with a Simple Control Structure”. In: *Computer Graphics Forum* vol. 35, no. 2 (2016), pp. 307–317 (cit. on p. 26).

[VSK17] Várady, T., Salvi, P., and Kovács, I. “Enhancement of a multi-sided Bézier surface representation”. In: *Computer Aided Geometric Design* vol. 55, no. Supplement C (2017), pp. 69–83 (cit. on p. 26).

[YZ04] Ying, L. and Zorin, D. “A Simple Manifold-based Construction of Surfaces of Arbitrary Smoothness”. In: *ACM Transactions on Graphics* vol. 23, no. 3 (Aug. 2004), pp. 271–275 (cit. on pp. 13, 19).

[Zan14] Zanaty, P. “Application of Generalized Expo-Rational B-splines in Computer Aided Design and Analysis”. PhD thesis. University of Oslo, 2014 (cit. on pp. 14, 19, 23).

[ZL17] Zhang, C. and Liu, L. “Manifold Construction Over Polyhedral Mesh”. In: *Communications in Mathematics and Statistics* vol. 5, no. 3 (Sept. 2017) (cit. on pp. 14, 25).

Papers

Paper I

Local refinement of GERBS surfaces with applications to interactive geometric modeling

Jostein Bratlie

[**PREPRINT**] Bratlie, J. "Local refinement of GERBS surfaces with applications to interactive geometric modeling". In: *39th International conference applications of mathematics in engineering and economics AMEE13*. Ed. by Pasheva, V. and Venkov, G. Vol. 1570. AIP Conference Proceedings. AIP Publishing, 2013, pp. 18–25

Abstract

Generalized Expo-rational B-splines (GERBS) is a blending spline construction, which blends local functions with up to C^k-smoothness. The blending spline combined with an ERB basis (ERBS curve, ERBS surface, and more) blends local functions with infinite smoothness. The ERBS blending construction was created especially with free form IGM in mind. One of the intrinsic properties of the blending spline constructions that help facilitate local editing is the minimal support of the basis functions. In this work, we look at local refinement of such blending spline tensor product surfaces, which could enhance its dynamic editing capabilities. We investigate knot insertion on blending spline surfaces and introduce a method for local refinement using blending.

I.1 Introduction

Expo-Rational B-Splines (ERBS) was first introduced in [LBD05] as an alternative to polynomial-based curves and surfaces for use in computer aided geometric design (CAGD) and then especially free-form editing. The ERBS basis functions and curve/surface constructions were later investigated; remarkably in [DLB06] and [Lak07], respectively. Later the ERBS was generalized, and the Generalized Expo-Rational B-Splines (GERBS)[DBL09] was introduced. With respect to IGM and traditional spline constructions, the GERBS construction differentiates itself by having local support without compromising the C^k smoothness introduced by the basis function. Furthermore, it has local

enrichment functions instead of coefficients which for IGM means the control-points have orientation. This gives the free-form modeler the possibility of editing the local functions by affine transformations, editing the parameters of a local refinement function or replacing one altogether.

An important feature of a spline framework for CAGD is refinement. Preferably geometry preservation such that it leaves the geometry unchanged before and after refinement. Among refinement methods for spline constructions are methods for knot insertion/removal, notably on B-splines, [Boe80; LM87]. Moreover, the B-spline construction has been extended into several other constructions with refinement properties, such as T-splines [Sed+03], PHT-splines [Den+08] or LR-splines [DLP13].

We look at local refinement with a particular focus on tensor-product surfaces. First, by taking a look at knot insertion on GERBS and then introducing a method for local refinement using blending; to preserve local support.

The blending spline construction is introduced in Chapter 2. In Section I.3, we investigate knot insertion on univariate blending splines with respect to preserving the construction's structural form, type, and local support. In Section I.4, we introduce a local refinement method for GERBS surface constructions that utilizes blending to preserve local support. Moreover, in Section I.5, we extend the refinement method with two methods for multi-layered refinement on a single interval. Finally, in Section I.6, we discuss the usages of these refinement methods and give some concluding remarks.

I.2 The blending spline construction

See Chapter 2.

I.3 Local refinement by knot insertion

Consider a blending spline on the interval $[t_k, t_{k+1}]$

$$f(t) = \ell_k(t) + (l_{k+1}(t) - \ell_k(t))\, \mathfrak{B}(t) \circ \omega_k, \tag{I.1}$$

where $\mathfrak{B}(t)$ is a B-function, ω_k is an affine mapping function from the global spline space to the space of the associated local function, ℓ_i. The knot interval is illustrated in Figure I.1. In the following theorem, we will consider knot insertion on a GERBS construction.

Theorem I.3.1. *Consider a blending spline, with a given type of local functions, on the interval* (t_k, t_{k+1})

$$f(t) = \ell_k(t) + (\ell_{k+1}(t) - \ell_k(t))\ \ \mathfrak{B} \circ \omega_k(t), \tag{I.2}$$

where $\omega_k(t) = \frac{t-t_k}{t_{k+1}-t_k}$ *is an affine mapping function from the global to the local function space. Inserting a new knot,* $\hat{t}$*, on the interval without modifying the existing local functions can be done while preserving the geometry but will not yield the same kind of construction.*

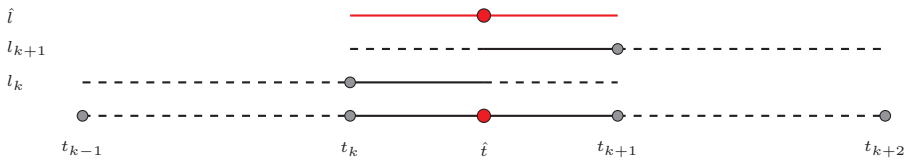


Figure I.1: Illustrates the cover for local functions ℓ_k and ℓ_{k+1} on a knot interval $[t_k, t_{k+1}]$ with a new knot $\hat{t}$ and an associated new local function $\hat{\ell}$.

Proof. Inserting a new knot in $[t_k, t_{k+1}]$ yields $[t_k, \hat{t}, t_{k+1}]$. We need to create an expression for the new associated local function $\hat{\ell}(t)$. For the first sub-interval, $(t_k, \hat{t})$, we have

$$f(t) = \ell_k(t) + \big(\hat{t} - \ell_k(t)\big) \;\; \mathfrak{B} \circ \omega_1(t), \tag{I.3}$$

where $\omega_1(t) = \frac{t-\hat{t}}{t_{k+1}-\hat{t}}$ is the affine mapping function on the new local sub-interval. Using (I.2) and (I.3)

$$\ell_k(t) + (\ell_{k+1}(t) - \ell_k(t)) \;\; \mathfrak{B} \circ \omega_k(t) = \ell_k(t) + \Big(\hat{\ell}(t) - \ell_k(t)\Big) \;\; \mathfrak{B} \circ \omega_1(t),$$

we can derive a new local function $\hat{\ell}(t)$ on the first sub-interval

$$\hat{\ell}(t) = \ell_k(t) + (\ell_{k+1}(t) - \ell_k(t)) \;\; \tfrac{\mathfrak{B} \circ \omega_k(t)}{\mathfrak{B} \circ \omega_1(t)}.$$

Looking at the other interval, $(\hat{t}, t_{k+1})$, we have

$$f(t) = \hat{\ell}(t) + \Big(\ell_{k+1}(t) - \hat{\ell}(t)\Big) \;\; \mathfrak{B} \circ \omega_2(t), \tag{I.4}$$

where $\omega_2(t) = \frac{t-\hat{t}}{t_{k+1}-\hat{t}}$ is the affine mapping function on the new local sub-interval. As for the first sub-interval we use (I.2) and (I.4)

$$\ell_k(t) + (\ell_{k+1}(t) - \ell_k(t)) \;\; \mathfrak{B} \circ \omega_k(t) = \hat{\ell}(t) + \Big(\ell_{k+1}(t) - \hat{\ell}(t)\Big) \;\; \mathfrak{B} \circ \omega_2(t),$$

and get an expression for the new local function $\hat{\ell}(t)$ on the second sub-interval

$$\hat{\ell}(t) = \ell_k(t) + (\ell_{k+1}(t) - \ell_k(t)) \, \tfrac{\mathfrak{B} \circ \omega_k(t) - \mathfrak{B} \circ \omega_2(t)}{1 - \mathfrak{B} \circ \omega_2(t)}.$$

On the whole interval, $[t_k, \hat{t}, t_{k+1}]$, the new local function looks as follows

$$\hat{\ell}(t) = \begin{cases} \ell_k(t) + (\ell_{k+1}(t) - \ell_k(t)) \;\; \tfrac{\mathfrak{B} \circ \omega_k(t)}{\mathfrak{B} \circ \omega_1(t)} & \text{if } t \in (t_k, \hat{t}), \\ \ell_k(t) + (\ell_{k+1}(t) - \ell_k(t)) \;\; \tfrac{\mathfrak{B} \circ \omega_k(t) - \mathfrak{B} \circ \omega_2(t)}{1 - \mathfrak{B} \circ \omega_2(t)} & \text{if } t \in (\hat{t}, t_{k+1}). \end{cases}$$

When looking at the new basis function, we note it is expressed in the form of a rational function and thus not of the same form as the original construction. ■

I.4 Refinement by blending

In this section, we will look at a local refinement technique to extend the blending construction to local refinement knots before we in the next section discuss two multi-layered extensions. Throughout the discussions, we will be using tensor-product blending spline surfaces.

Definition I.4.1 (Refinement knot)**.** Given a tensor-product blending spline surface with $n_u \times n_v$ local surfaces. The knot vectors $\vec{u} = \{u_i\}_{i=0}^{n_u+1}$ and $\vec{v} = \{v_i\}_{i=0}^{n_v+1}$, and associated local surfaces $\bar{L}_{i,j}$, $i = 1, \ldots, n_u$, $j = 1, \ldots, n_v$. We define a refinement knot $r = [r_u, r_v]$, $r_u \in (u_0, u_{n_u+1})$, $r_u \notin \vec{u}$, $r_v \in (v_0, v_{n_v+1})$, $r_v \notin \vec{v}$ and a refinement surface s_r.

On the refinement interval $[u_i, u_{i+1}], [v_j, v_{j+1}]$, illustrated in Figure I.2, where $u_i < r_u < u_{i+1}$ and $v_j < r_v < v_{j+1}$, the refined blending spline patch can be expressed as follows

$$S = G + (S_r - G)\ A \circ \lambda(r), \tag{I.5}$$

where $A \circ \lambda(r) = \mathfrak{B} \circ \omega_{\vec{u}}\ (r_u)\ \mathfrak{B} \circ \omega_{\vec{v}}\ (r_v)$ is a custom tensor-product blending spline basis and G is the original blending spline patch on the interval.

I.4.1 Geometry preservation

If refining an existing model for the purpose of adding details to a localized area or interval, preserving the geometry, such that the global geometry is the same before and after refinement is crucial (e.g., a 3D artist creating models for video games). We can extend the blending technique to conform with geometry preservation by creating a refinement patch that is identical to the underlying blending spline patch on the refinement interval. Using (I.5) while setting S_r equal to the blending spline surface patch G on the refinement interval

$$S = G + (S_r - G)\ A \circ \lambda(r) \Rightarrow S = G + (G - G)\ A \circ \lambda(r) = G, \tag{I.6}$$

we can clearly see that this will yield the same global geometry as before refinement.

I.4.1.1 Constructing the refinement patch

On a tensor-product blending spline surface, the local refinement interval spans two knots in both parametric directions. This gives us a blending spline patch with 2×2 local surfaces. We use copies of the corresponding local surfaces from the original blending spline surface as initial local surfaces. Additionally, we need to utilize the same type of B-function.

Finally, for each of the refinement patch's local surfaces, if the local surface is interior in the original surface, for a given parametric direction, the parametric domain must be shifted according to the original knot vector as follows

$$\rho(t) = \begin{cases} t_k + (t - t_k)\,\frac{\hat{t} - t_k}{t_{k+1} - t_k}, & \text{if local surface belonged to } t_k, \\ \hat{t} + (t - t_k)\,\frac{t_{k+1} - \hat{t}}{t_{k+1} - t_k}, & \text{if local surface belonged to } t_{k+1}. \end{cases} \tag{I.7}$$

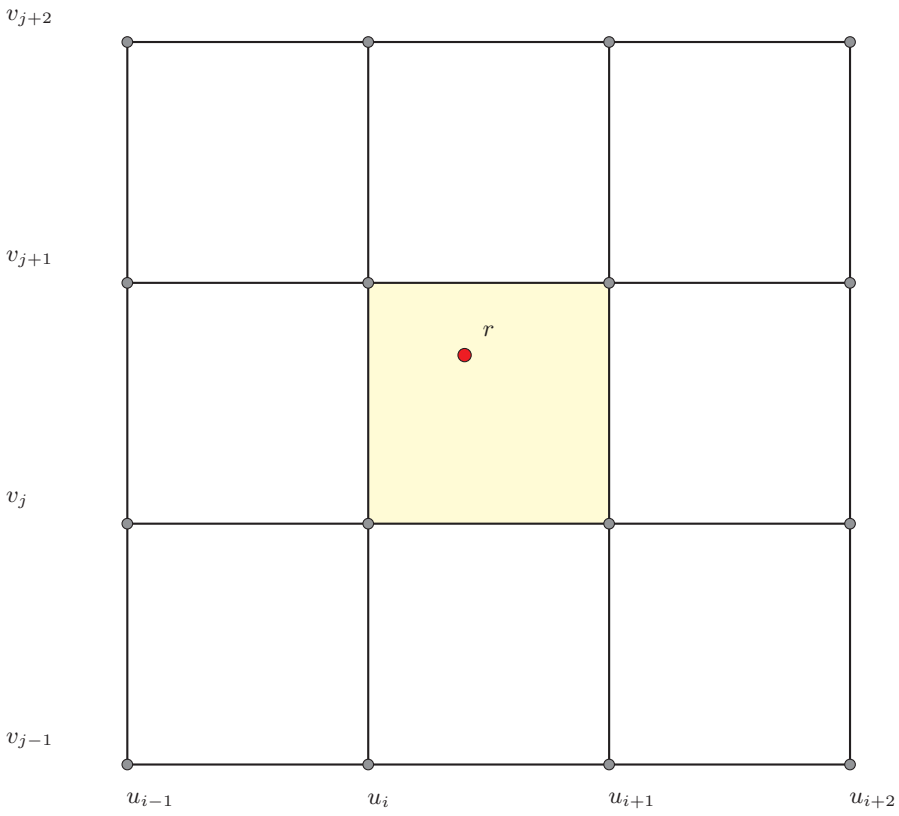


Figure I.2: The figure illustrates a refinement point r on the refinement interval of a tensor-product blending spline surface.

I.5 Multi-layer blending

When refining an interval on the blending spline, we add a refinement knot, which acts as a local interpolation knot in the refined interval. This section describes two ways of extending this refinement technique to allow for more than one layer of refinement on one refinement interval. The first method adds multiple refinement knots to one refinement interval. The second method is a multi-level method where each refinement patch is a new blending spline patch. Illustrated in Figures I.3 and I.4.

I.5.1 Multi-knot blending spline patch refinement

We want to extend the blending technique to allow for multiple refinement knots on one refinement interval. We define a refinement knot vector $[r_k]_{k=0}^{\gamma-1}$ on the refinement interval, where γ is the number of refinement knots, and each

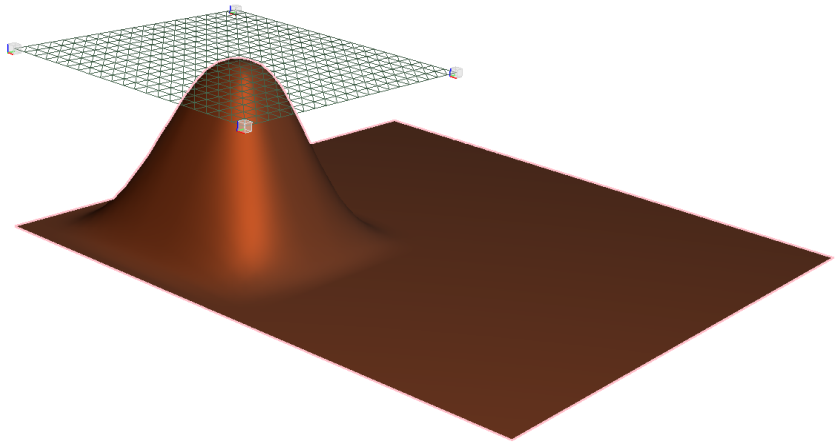


Figure I.3: The figure shows a single-level refinement of a tensor-product blending spline surface.

refinement knot $r_k = [r_{k,u}, r_{k,v}]$ fulfills the requirements defined in Definition I.4.1. Each knot has an accompanying refinement patch, $s_{r,k}$. Extending (I.5), we can express the refined blending spline on the refined interval recursively as follows

$$\begin{aligned} S_n &= S_{n-1} + (S_{r,n} - S_{n-1})\ A \circ \lambda(r_n), \\ S_0 &= G + (S_{r,0} - G)\ A \circ \lambda(r_0), \end{aligned} \tag{I.8}$$

where $S_{r,n}$ is the refinement patch associated with layer $n \in \mathbb{N}^0$. Observing the expression we note that the n-th refinement knot, $A \circ \lambda(r_n) = 1$, interpolates the global geometry

$$S_n = S_{n-1} + S_{r,n} - S_{n-1} = S_{r,n}.$$

If all local refinement patches fulfils the requirements described in Section I.4.1 we see that by extending (I.5) using (I.8) to several layers we get the same result for multi-layered blending as for a single refinement knot in (I.6). For the case of $S_{n=2}$ we have

$$\begin{aligned} S_2 &= G + (S_{r,2} - S_{r,1})\,\alpha + (S_{r,1} - S_{r,0})\,\alpha^2 + (S_{r,0} - G)\,\alpha^3 \\ &\Rightarrow G + (G - G)\,\alpha + (G - G)\,\alpha^2 + (G - G)\,\alpha^3 = G, \end{aligned} \tag{I.9}$$

where $\alpha^n = \prod_{i=1}^{n} A \circ \lambda(r_{n-i})$.

I.5.2 Multi-level blending spline patch refinement

The second method for multi-layered blending is a multi-level method. Instead of allowing multiple refinement knots on one refinement interval, we apply the

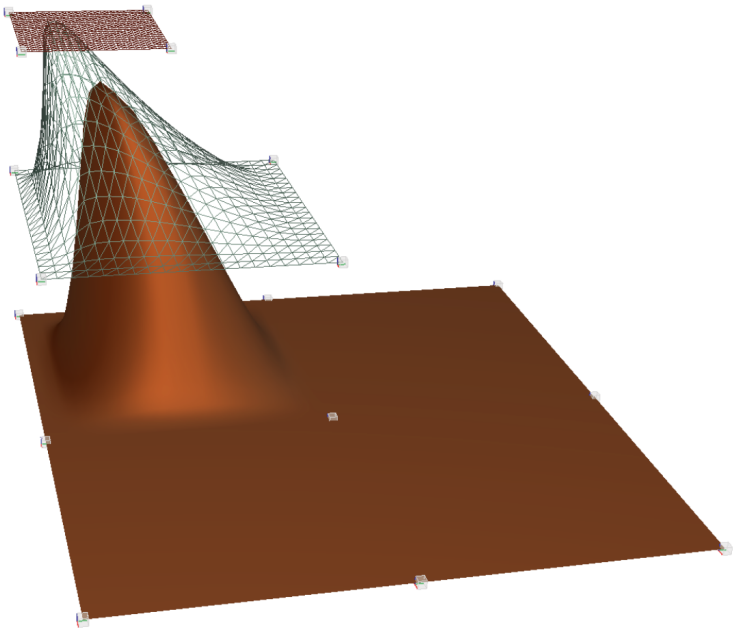


Figure I.4: The figure shows a two-level refinement of a different tensor-product blending spline surface, where the local surfaces have been edited by affine transformations; after refinement.

refinement process over again on the new refinement patch. This, of course, gives that the refinement patch must be a blending spline tensor-product surface. The final refinement patch can then be expressed, recursively, as follows

$$S_n = G_n + (S_{n-1} - G_n)\ A \circ \lambda(\hat{r}_n),$$
$$S_0 = G_0 + (S_{r,0} - G_0)\ A \circ \lambda(\hat{r}_0),$$

where G_n is the blending spline patch on the refinement interval for a given level, $\hat{r}_n$ is the local refinement knot for the blending spline surface refinement interval on that level, and $S_{r,0}$ is the refinement patch on the inner-most level. We note that for a given level $n \in \mathbb{N}^0$, the refined blending spline patch, S_n, is the refinement patch on the higher level, $S_{r,n+1}$. In contrast to the first multi-layer method, looking at the expression for S_n, we observe that the refinement patch S_{n-1} will, at the refinement knot $A \circ \lambda(\hat{r}_n) = 1$, interpolate the higher-level geometry

$$S_n = G_n + S_{n-1} - G_n = S_{n-1}.$$

As for the multi-knot method, we can show that, if the refinement patches on each level fulfil the requirements described in Section I.4.1, we have for this method, with respect to G_n, as we have in (I.6) and (I.9). For the case of $S_{n=2}$, we have

$$\begin{aligned} S_2 &= G_2 + ((G_1 + ((G_0 + (S_{r,0} - G_0)\ A \circ \lambda(\hat{r}_0)) - G_1)\ A \circ \lambda(\hat{r}_1) - G_2)\ A \circ \lambda(\hat{r}_2) \\ &\Rightarrow G_2 + ((G_2 + ((G_2 + (G_2 - G_2)\ A \circ \lambda(\hat{r}_0)) - G_2)\ A \circ \lambda(\hat{r}_1) - G_2)\ A \circ \lambda(\hat{r}_2) \\ &= G_2. \end{aligned} \tag{I.10}$$

Remark I.5.1 (Remark: multi-level vs. multi-layer). Where for the multi-layer method $r_k \neq r_{k+1}$, this is not true for the multi-level method as the refinement knots, $\hat{r}_k$ belong to independent refinement patches. Moreover, this means that if $\hat{r}_0 = \hat{r}_1 = \cdots = \hat{r}_n$ when $A \circ \lambda(\hat{r}_n) = 1$, the inner-most refinement patch will interpolate the global geometry.

I.6 Concluding remarks

Knot insertion on a blending spline for a given combination of basis-functions and associated local functions, where we preserve local support, can not result in a construction of the same algebraic form. With this in mind, we introduced a local refinement method based on blending, which, with given constraints on the refinement patch, produces a locally refined blending spline surface that preserves the global geometry.

The two multi-layer extensions of the proposed blending method provide a simple way to do further refinement. The multi-knot method provides a layered refinement by adding a new interpolating layer of detail on top of the existing construction. The multi-level method is a method that gives the ability to refine inwards by not adding an interpolating layer on top but adding a layer underneath the construction. A combination of the two methods will then give the combined control of detailed refinement.

In this work, we have focused on geometry preservation of the blending spline construction. A natural expansion is to look into how well the blending refinement construction would perform when using approximated local functions.

References

[Boe80] Boehm, W. “Inserting New Knots into B-spline Curves”. In: *Computer Aided Design* vol. 12, no. 4 (1980), pp. 199–201 (cit. on p. 72).

[Bra13] Bratlie, J. “Local refinement of GERBS surfaces with applications to interactive geometric modeling”. In: *39th International conference applications of mathematics in engineering and economics AMEE13*. Ed. by Pasheva, V. and Venkov, G. Vol. 1570. AIP Conference Proceedings. AIP Publishing, 2013, pp. 18–25 (cit. on p. 71).

[DBL09] Dechevsky, L. T., Bang, B., and Lakså, A. “Generalized Expo-Rational B-Splines”. In: *International Journal of Pure and Applied Mathematics* vol. 57, no. 6 (2009), pp. 833–872 (cit. on p. 71).

[DLB06] Dechevsky, L. T., Lakså, A., and Bang, B. “Expo-Rational B-Splines”. In: *International Journal of Pure and Applied Mathematics* vol. 27, no. 3 (2006), pp. 319–362 (cit. on p. 71).

[Den+08] Deng, J., Chen, F., Li, X., Hu, C., Tong, W., Yang, Z., and Feng, Y. “Polynomial splines over hierarchical T-meshes”. In: *Graphical Models* vol. 70, no. 4 (2008), pp. 76–86 (cit. on p. 72).

[DLP13] Dokken, T., Lyche, T., and Pettersen, K. F. “Polynomial splines over locally refined box-partitions”. In: *Computer Aided Geometric Design* vol. 30, no. 3 (2013), pp. 331–356 (cit. on p. 72).

[Lak07] Lakså, A. “Basic properties of Expo-Rational B-splines and practical use in Computer Aided Geometric Design”. (Dr.philos.) PhD thesis. University of Oslo, 2007 (cit. on p. 71).

[LBD05] Lakså, A., Bang, B., and Dechevsky, L. T. “Exploring Expo-Rational B-splines for Curves and Surfaces”. In: *Mathematical methods for Curves and Surfaces*. Ed. by Dæhlen, M., Mørken, K., and Schumaker, L. L. Nashboro Press, 2005, pp. 253–262 (cit. on p. 71).

[LM87] Lyche, T. and Mørken, K. “Knot removal for parametric B-spline curves and surfaces”. In: *Computer Aided Geometric Design* vol. 4, no. 3 (1987), pp. 217–230 (cit. on p. 72).

[Sed+03] Sederberg, T. W., Zheng, J., Bakenov, A., and Nasri, A. “T-splines and T-NURCCs”. In: *ACM SIGGRAPH 2003 Papers*. SIGGRAPH ’03. San Diego, California: ACM, 2003, pp. 477–484 (cit. on p. 72).

Paper II

Fitting of discrete data with GERBS

Jostein Bratlie, Rune Dalmo, Peter Zanaty

[**PREPRINT**] Bratlie, J., Dalmo, R., and Zanaty, P. "Fitting of Discrete Data with GERBS". in: *Large-Scale Scientific Computing 2013*. Ed. by Lirkov, I., Margenov, S., and Waśniewski, J. Vol. 8353. Lecture Notes in Computer Science. Springer, 2014, pp. 569–576

II

Abstract

In this paper, we present a study of fitting discrete data with blending splines.We investigate different ways to determine interpolation knots and generate local curves by partitioning the parametric space and solving a corresponding least-squares fitting problem. We apply our technique to discrete evaluations of continuous synthetic benchmark functions and compare the resulting blending spline to the original data with respect to errors and performance.

II.1 Introduction

In this work, we investigate the properties of fitting blending splines [DBL09] to regular discretized data. In blending spline constructions, local functional coefficients are blended by blending spline basis functions. The choice of basis functions and the local enrichment functions determine the local and hence the global approximation properties of the resulting space.

One of the intrinsic properties of the blending spline bases are the minimal support of the basis functions, which allows for a simple approximation technique; instead of storing the individual data points and then blending the corresponding local functions together, node by node, we can choose the interpolation knots and the accompanying local functions freely, depending on the data itself.

Using this, we investigate various techniques to partition the parametric space of the blending spline across the discrete data by changing the interpolation knots and simultaneously adjusting the corresponding coefficient functions. In addition, we look at the performance of the different constructions with respect to approximation.

Many papers have been published on the topic of data fitting, data reduction, compression, and smoothing with B-splines using various methods. We mention

here the knot removal technique presented by Lyche and Mørken in [LM87] and with a different approach by Eck and Hadenfeld in [EH94], and the shape-preserving knot removal method by Schumaker and Stanley in [SS96]. We also mention the work done by Saux and Daniel in [SD98; SD99] on estimating criteria for fitting and data reduction of polygonal curves using B-splines. We leave these topics for now and focus on a few simple methods for constructing blending spline local functions.

In Section II.2, we start by giving a brief introduction to blending spline and its construction and the partitioning and fitting setup we use throughout the paper. Then in Section II.3, we describe the different partitioning algorithms, and then the description of the fitting-method follows in Section II.4. Finally, in Section II.5, we give some concluding remarks where we discuss our findings and future work.

II.2 Preliminaries

The blending spline construction is defined in Chapter 2. The local functions ℓ_i throughout this paper shall be Taylor expanding polynomials up to a multiplicity μ_i

$$\ell_i(t-t_i) = \sum_{j=0}^{\mu_i} c_{i,j} \frac{(t-t_i)^j}{j!}, \tag{II.1}$$

and the corresponding blending spline base $B_i(t)$ is required to have vanishing derivatives of order up to, including, μ_i. For the rest of the paper we will use the ERBS basis described in [LBD05], which is capable of transfinite Hermite interpolation, i.e., all of its derivatives vanish at all knots.

The knots $(\vec{t})$ and the multiplicities $(\vec{\mu})$ together define a spline space, where the coefficients $(\vec{c})$ have a natural meaning corresponding to a Hermite interpolation problem.

II.2.1 Partitioning and fitting

In digital systems, we often have to deal with continuous (analog) input data. The way it is handled is that the analog signal is converted to digital by sampling and quantizing (digitizing), and the resulting raw digital data is being used instead of the original data. Often it is a requirement to produce outputs that are either continuous or sampled at a higher rate than that of the input data; this problem translates into interpolation/extrapolation or approximation problems depending on the requirements.

In the current paper, we are interested in comparing different strategies for representing uniformly sampled uni-variate functions using blending spline based approximation, see Figure II.1. The partitioning algorithms work on a sampled data of two benchmark functions, given by

$$f_1(t) = \begin{pmatrix} \ln(t+1) \\ -t\sin(2t+1)) \end{pmatrix}, \qquad t \in [0,1],$$

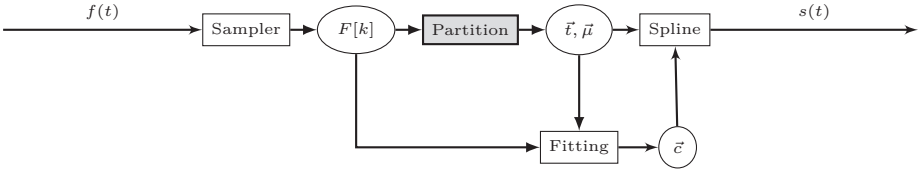


Figure II.1: Block diagram showing how a continuous signal is discretized and represented as a spline via partitioning and fitting of the data.

$$f_2(t) = \begin{pmatrix} t \\ t\sin(\frac{1}{t}) \end{pmatrix}, \qquad t \in [0.01, 0.5],$$

and their task is to select the knot configuration and the corresponding local multiplicities of the spline space.

Then a fitting algorithm obtains the coefficients to the spline representation; finally, we compare the resulting splines with both the original continuous benchmark functions $f_1(t)$, $f_2(t)$ and their sampled discrete versions $F_1[k]$, $F_2[k]$ and discuss some properties of the resulting transformations.

II.3 Partitioning algorithms

In order to fit a blending spline construction (see Section 2.4) to discrete data, it is necessary to decide where to place the interpolation knots ($\vec{t}$) and to decide the corresponding multiplicities ($\vec{\mu}$) of the local functions. This can be done in a number of different ways. We describe three different algorithms for constructing local curves.

II.3.1 Uniform partitioning

As a starting point with uniform sampling, we define a *knot* for each discrete data point in $F[k]$. Next, the number of knots is reduced by selecting a subset of F in order to define the spline space. We add as a note here that the data is assumed to be appropriate for selection. (In some cases, it is common to smooth the data before selecting to reduce errors or avoid problems related to oscillation.) It is possible to increase the degree by selecting derivatives for each knot. We illustrate uniform partitioning with three different examples

1. fixed sample rate
2. a specified number of knots, and
3. parametric stride.

In the first case, the sample rate simply states how many knots to skip between the selected knots. Hence, a sample rate of two selects every second knot, whereas a sample rate of 10 selects every 10th knot.

The number of knots in the second case defines the size of the resulting knot vector. This implies a computation of the sampling rate depending on the number of elements in F.

We consider parametric stride, where we select knots equidistant in parametric space, in the current paper.

II.3.2 Curvature based partitioning

Moving away from uniform partitioning, we describe in brief a naive, curvature extrema-based partitioning approach. From the discrete function $F[k]$, $k = 1, \ldots, M$, we compute for each interior knot t_i, $i = 2, \ldots, M-1$ the radius of the circumscribed circle of triangle $R[i] = R_{circ}\triangle(F[i-1], F[i], F[i+1])$. These values correspond to the curvature of the curve at the corresponding interior points. Next, we select the extrema of these values as they, together with the two endpoints, constitute the points of interest (for more details on feature point selection consult [SD99; Tha89]).

To be able to scale the method, the resulting set of feature points is processed further. Feature points that are too close are filtered out, and new feature points are introduced uniformly between feature points that were too far away.

II.3.3 Partitioning based on inflexion

We look at two different approaches based on relative angular changes in the discrete data set. In both cases, we consider the angle between the two vectors spanning a sample point. Where we in the first approach consider the change in the angle by tracing the curve, we start by sorting the angles into different buckets in the other.

II.3.3.1 Inline traced partitioning:

In the first variation, we look at an approach where we consider the linear interpolation between two neighboring data points to be a vector that provides a first derivative in one point. Given $\vec{a}$ and $\vec{b}$, two vectors, we use the dot product between vectors and the angular difference γ of $\vec{a}$ and $\vec{b}$

$$\cos(\gamma) = \frac{< \vec{a}, \vec{b} >}{|\vec{a}| \left|\vec{b}\right|}. \tag{II.2}$$

Given the discrete data set $F[k]$ and an empty set $\mathbf{q}$ to store the detected feature points. Apply (II.2) to the vectors $\mathbf{a} = p_1 - p_0$ and let $\mathbf{b}$ "run" along the curve, starting with $\mathbf{b} = p_2 - p_1$, then $\mathbf{b} = p_3 - p_2$, and so on. By comparing the results of applying (II.2), whenever the sign of the gradient of the resulting "curve" changes, we find a point of inflection on the curve given by linear interpolation between points in $\mathbf{p}$.

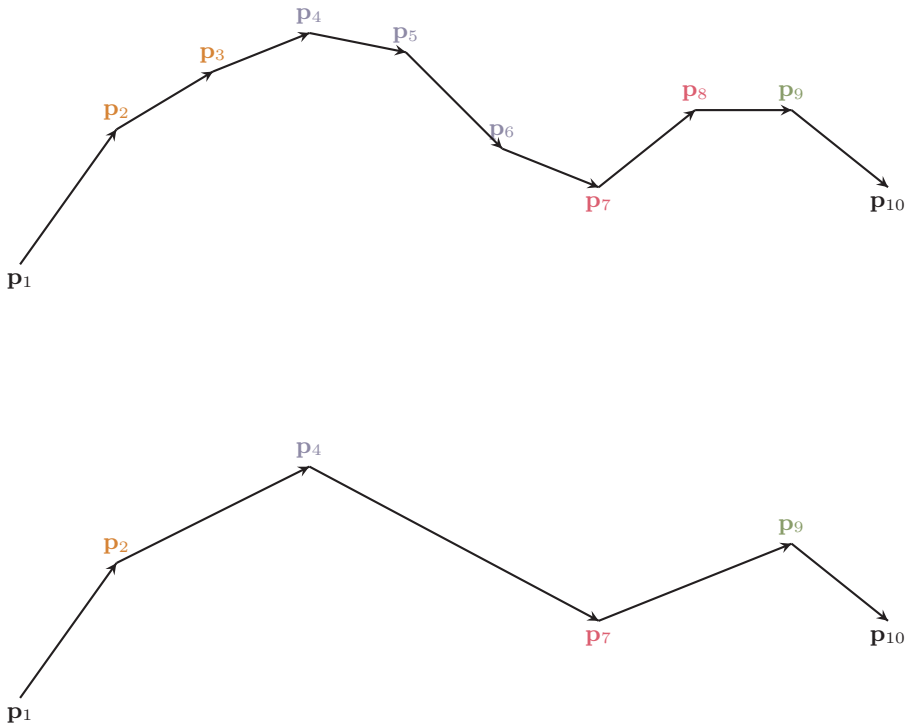


Figure II.2: Ten points and the linear interpolation in-between drawn as vectors. Top: The ten sample points. Interior sample points sorted into three different buckets. Bottom: The interior feature points are kept after partitioning.

II.3.3.2 Bucket based partitioning:

The second approach is to do an angular difference based partitioning by segmenting knots into buckets, each bucket corresponding to a range subset of possible angular differences between the forward and the backward edge. From the discrete function $F[k]$, $k = 1, \ldots, M-1$, we compute for each interior knot the angle between the two adjacent vectors, $\vec{a}$ and $\vec{b}$, where $a = F[k+1] - F[k]$ and $b = F[k+2] - F[k+1]$. We sort the angles and divide the knots into equal sized-buckets; this can be seen in Figure II.2. Next, we run along the curve selecting feature knots, where if the following knots belong to the same bucket, only the first is kept as a feature knot.

In addition to the found interior knots, the end-point knots are also kept as feature knots. Finally, the resulting set of feature points are processed further; feature points that are too close are filtered out.

II.4 Fitting

Once the interpolation space is set up, by defining the knot vectors ($\vec{t}$) and the multiplicities ($\vec{\mu}$), the problem of finding the coefficients to best match the given discrete data set $F[k], k = \{k_1, k_2, \ldots, k_M\}$ of M points remains. For this purpose, we will use the best L_2 approximant, given by the coefficient minimizing the mean squared errors

$$\|S - F\|^2 = \sum_{i=1}^{M} |s(k_i) - F[k_i]|^2.$$

The coefficients are obtained by the usual technique of solving least squares problems. We restrict our investigation to cases where the fitting problem is not ill-posed.

Figures II.3 and II.4 shows the performance of the considered methods introduced in Section II.3. The x-axis shows the percentage of the original data that is being used, while the y-axis displays the signal-to-noise ratio (SNR) measured in dB, defined as

$$\mathrm{SNR} = 10 \log_{10} \left(\frac{\|F\|^2}{\|F - S\|^2} \right),$$

where F stands for the original discrete data, and S represents the reconstructed data, and $\| \cdot \|^2$ stands for the square of the usual L_2 norm.

II.5 Concluding remarks

Our technique is local; hence there is a small bandwidth in the resulting matrix in the least squares fitting, which in turn gives a computational advantage over global methods, i.e., classical polynomial B-splines.

The computational complexity of each feature detection method is linear and readily parallelizable; similarly since the splines are local, the reconstruction and evaluation can also be performed in parallel.

We note a trade-off between smoothing and interpolation which can be adjusted depending on how confident we are in the data and the condition number of the fitting. Furthermore, the smoothness of the resulting curve can be easily adjusted, e.g., to fulfill a smoothness criteria of the underlying physics of the discrete sample points.

The primary utilization is to reduce an original data set and use a blending spline to represent the final data. From the two synthetic benchmarks, we can conclude that the two types of feature extraction coupled with the coloring or the refining extension allowed for the construction of a series of tune-able spline spaces which performed at least as good as the least-squares fitting for smooth inputs and proved to be much more stable for oscillating irregular input data.

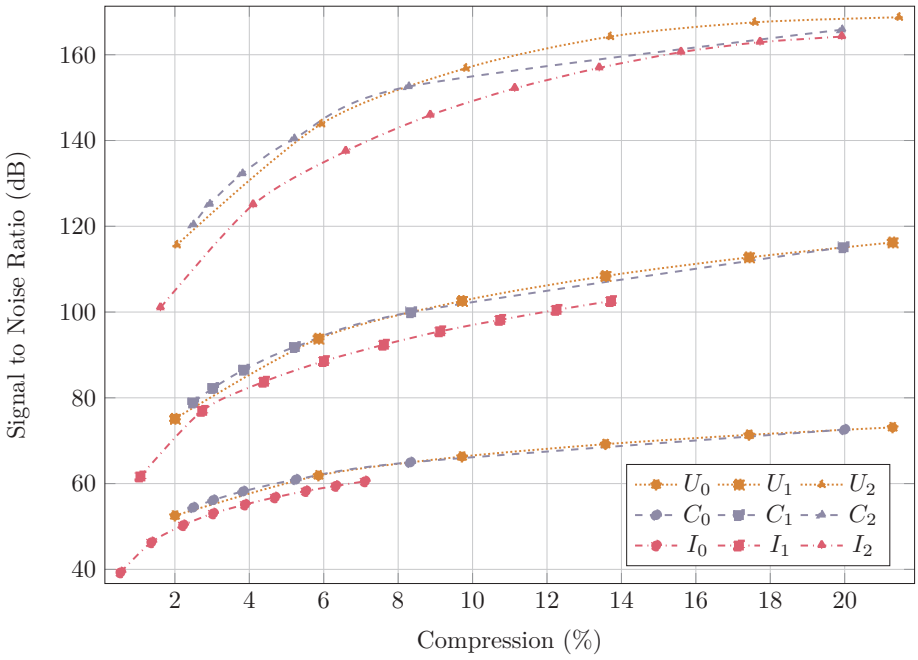


Figure II.3: Error rates for the smooth synthetic benchmarks. The teal lines U_0,U_1,U_2 represent the uniform algorithm, the purple lines C_0,C_1,C_2 stand for the curvature-based refining method, while the orange marks I_0,I_1,I_2 show the performance of the bucketing algorithm based on angles. The lower indices correspond to the applied uniform multiplicity, μ_i in (II.1).

II.5.1 Future work

Future work related to applications includes extending the current study to applications in cartography and animation data, including adapting the presented ideas to industry standard representations used there, i.e., Catmull-Rom splines in animations and Bézier curves in cartography.

The locality of the method makes it a suitable candidate for streaming data, one particular potential area of use can be in massive multiplayer online (MMO) games within the computer games industry, where large amounts of data of similar structure have to be handled in real-time. Transferring data over a limited bandwidth, especially for relatively sizeable discrete data sets, translates to simply transferring coefficients via the networks since the coefficients alone are enough to reconstruct data from a sender on the receiver's end.

Finally, to put a last note for future work, we believe more sophisticated methods for partitioning the parametric space could further enhance the results. It could be interesting to apply well-studied principles for data reduction, such as (shape-preserving) knot removal or those based on features and criteria of

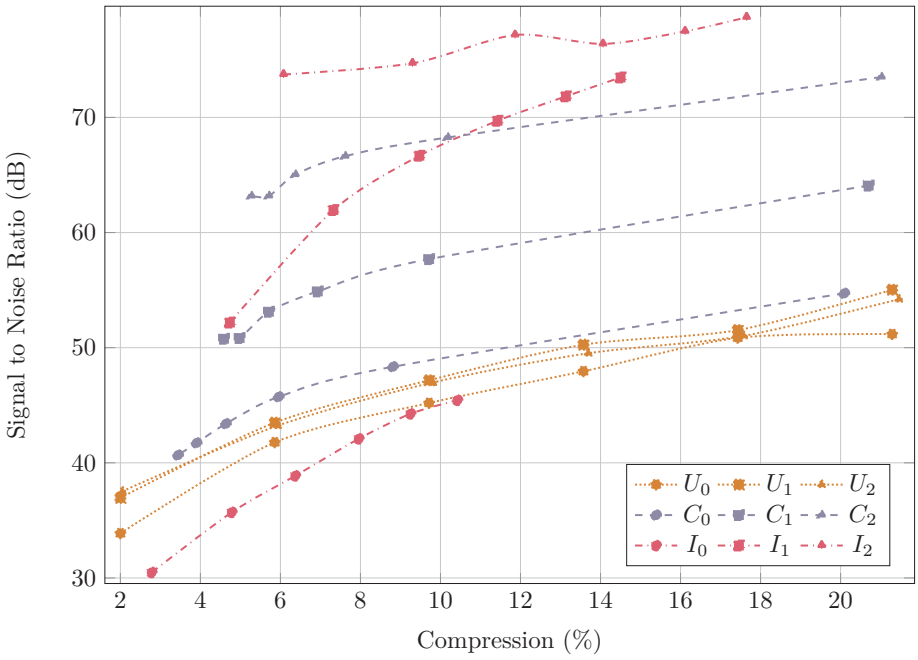


Figure II.4: Error rates for the oscillating synthetic benchmarks. The teal lines U_0,U_1,U_2 represent the uniform algorithm, the purple lines C_0,C_1,C_2 stand for the curvature-based refining method, while the orange marks I_0,I_1,I_2 show the performance of the bucketing algorithm based on angles. The lower indices correspond to the applied uniform multiplicity, μ_i in (II.1).

the original data.

References

[BDZ14] Bratlie, J., Dalmo, R., and Zanaty, P. "Fitting of Discrete Data with GERBS". In: *Large-Scale Scientific Computing 2013*. Ed. by Lirkov, I., Margenov, S., and Waśniewski, J. Vol. 8353. Lecture Notes in Computer Science. Springer, 2014, pp. 569–576 (cit. on p. 81).

[DBL09] Dechevsky, L. T., Bang, B., and Lakså, A. "Generalized Expo-Rational B-Splines". In: *International Journal of Pure and Applied Mathematics* vol. 57, no. 6 (2009), pp. 833–872 (cit. on p. 81).

[EH94] Eck, M. and Hadenfeld, J. "Knot removal for B-spline curves". In: *Computer Aided Geometric Design* vol. 12, no. 3 (1994), pp. 259–282 (cit. on p. 82).

[LBD05] Lakså, A., Bang, B., and Dechevsky, L. T. "Exploring Expo-Rational B-splines for Curves and Surfaces". In: *Mathematical methods for Curves and Surfaces*. Ed. by Dæhlen, M., Mørken, K., and Schumaker, L. L. Nashboro Press, 2005, pp. 253–262 (cit. on p. 82).

[LM87] Lyche, T. and Mørken, K. "Knot removal for parametric B-spline curves and surfaces". In: *Computer Aided Geometric Design* vol. 4, no. 3 (1987), pp. 217–230 (cit. on p. 82).

[SD98] Saux, E. and Daniel, M. "Estimating Criteria for Fitting B-spline Curves: Application to Data Compression". In: *Proceedings of GraphiCon '98*. Ed. by Klimenko, S. and Shikin, E. Moscow State University, 1998 (cit. on p. 82).

[SD99] Saux, E. and Daniel, M. "Data reduction of polygonal curves using B-splines". In: *Computer Aided Design* vol. 31, no. 8 (1999), pp. 507–515 (cit. on pp. 82, 84).

[SS96] Schumaker, L. L. and Stanley, S. S. "Shape-preserving knot removal". In: *Computer Aided Geometric Design* vol. 13, no. 9 (1996), pp. 851–872 (cit. on p. 82).

[Tha89] Thapa, K. "Data compression and critical points detection using normalized symmetric scattered matrix". In: *Proceedings of Auto Carto 9*. Baltimore, Maryland, USA, 1989, pp. 78–89 (cit. on p. 84).

Paper III

Evaluation of smooth spline blending surfaces using GPU

Jostein Bratlie, Rune Dalmo, Børre Bang

[**PREPRINT**] Bratlie, J., Dalmo, R., and Bang, B. "Evaluation of smooth spline blending surfaces using GPU". in: *Curves and surfaces. 8th International Conference*. Ed. by Boissonnat, J.-D., Cohen, A., Gibaru, O., Gout, C., Lyche, T., Mazure, M.-L., and Schumaker, L. L. Vol. 9213. Lecture Notes in Computer Science. Springer, 2015, pp. 60–69

Abstract

Recent development in several aspects of research on blending splines has opened up new application areas. We propose a method for evaluation and rendering of smooth blending type spline constructions using the tessellation shader steps of modern graphics hardware. In this preliminary study, we focus on concepts and terminology rather than implementation details. Our approach could lead to more efficient, dynamic, and stable blending-type spline based applications in fields such as interactive modeling, computer games, and more.

III.1 Introduction

The purpose of this paper is to introduce a concept for evaluation and rendering blending spline constructions, surfaces, in particular, using features available in recent versions of modern rendering pipelines [Sch+14], most notably the OpenGL [SA10], maintained by the Khronos group and Microsoft® DirectX® (DirectX) [Mic09]. Since the year 2000 the R&D group Simulations at UiT Narvik, conducted research on blending splines through publications such as expo-rational B-splines (ERBS) [LBD05] and, generalized expo-rational B-splines (GERBS) [DBL09; DLB06]. Blending splines enjoy some properties, including Hermite interpolation at the knots [Lak07] and minimal support combined with C^k-smooth basis functions, making them attractive to interactive geometric modeling and smooth representations of parametric curves and surfaces.

Despite the flexibility of the construction, there is a price to pay, particularly with respect to interactive geometric modeling, due to the cost of the spline basis function evaluator. The performance is constrained by the following limitations:

1. evaluation of the ERBS basis requires an integration step, and

2. the graphics rendering hardware is designed to support triangle constructions and simple cases of classic splines on Bézier form, i.e., mapped to the interval $[0, 1]$.

The first limitation was addressed by Zanaty in [Zan14], where a relationship between the ERBS basis function and Sigmoidal functions was explored. The second limitation is addressed in this paper.

Recently in [Lak13], the blending splines were expressed in terms of linear B-splines. This is interesting since the rendering pipelines mentioned above were designed to be used with ordinary B-splines.

In this work, we consider tessellation techniques which are now standardized across vendor-specific APIs. Therefore, it is possible to adapt and exploit such tessellation steps to obtain rendering methods applicable to extended B-spline constructions. We seek to describe the relevant technology: blending splines and rendering pipelines and the concepts necessary for evaluation and rendering.

In the following sections, we describe blending splines, as in [Lak13], followed by an overview of the relevant steps of the graphics pipeline present in modern GPU hardware. Next, we introduce and define the critical components of the proposed rendering- and evaluation method, followed by a description of the method itself. Finally, we give our concluding remarks, discuss some theoretical performance results, and suggest topics for future work.

III.2 Spline blending functions

See Chapter 2.

III.3 GPU Tessellation

The most recently added shader component in the GPU rendering pipeline is the tessellation shader. It consists of three sub-components, steps two through four, between the vertex shader and the geometry shader, as shown in Figure III.1.

The tessellation shader steps operate on a type of shader primitive called *patch*. The shader patch primitive can be of three different types; isoline, triangle, or quad. In this work, we are primarily interested in the quad patch type primitive, as it fits the shape of the tensor-product parametric domain.

The tessellation step is controlled through three different substeps; control, tessellation and evaluation. The control and evaluation substeps are programmable, while the tessellation substep is hardware implementation-specific. In the following, we provide a brief description of each substep of the tessellator.

> *Control*; specifies which type of patch primitive to be considered and the amount of tessellation applied to each patch. It provides control of tessellation inside the patch and on the boundary of the patch independently.

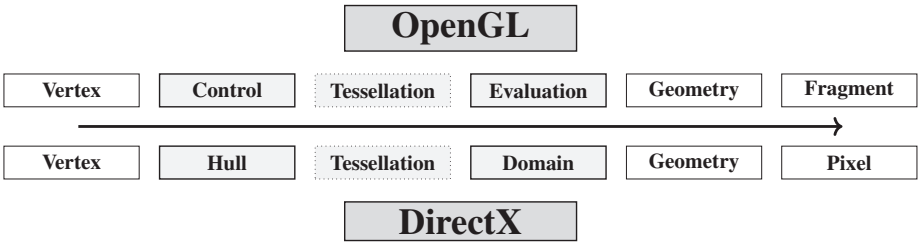


Figure III.1: The three tessellation shader steps are shown as parts of the OpenGL and DirectX rendering pipelines. The control- and evaluation shaders, using OpenGL's terminology, illustrated as shaded gray boxes, are programmable. The primitive generator, or tessellation step, is fixed.

Tessellation; which is not programmable, only controllable, performs the actual tessellation. It generates primitives. We can think of this step as to where the topology of the tessellation is induced.

Evaluation; determines the position of the new tessellated vertex. It is based on affine transformations and performs in a manner similar to what the vertex shader does for a vertex when combined with a basic primitive, such as point, line, or triangle-strip.

A vertex generated by the tessellation step has a normalized position within the tessellation patch. This means that the evaluation step works on coordinates on the range $[0, 1]$ and, in the case of triangle type patch primitives, barycentric coordinates.

By considering the blending spline construction described in Chapter 2 we propose building an evaluator based on the tessellation steps. Then, the tessellation patches of type line, triangle, and quad, provided by the control step, take the roles as a render "patch" on a blending type spline curve, spline triangle surface, and spline tensor-product surface, respectively. As we shall see in the following section, this is one layer in a hierarchical blending construction.

We conclude this section by briefly mentioning the roles of the other steps, tessellation, and evaluation. The tessellator determines parameter values stating wherein the parametric domain a blending type curve or surface is to be evaluated. The tessellation evaluator is a shader-program implementation of a blending spline evaluator.

III.4 Render-lattice, -patch and -loci

In [Lak13], a concept for an ERBS-construction on irregular grids was presented. The concept is to divide spline knot nets into regular and irregular grids, leaving us with three different types of points at the knots; regular-, T- and star points, as shown in Figure III.2. In [Lak13] T- and star-points are defined as follows

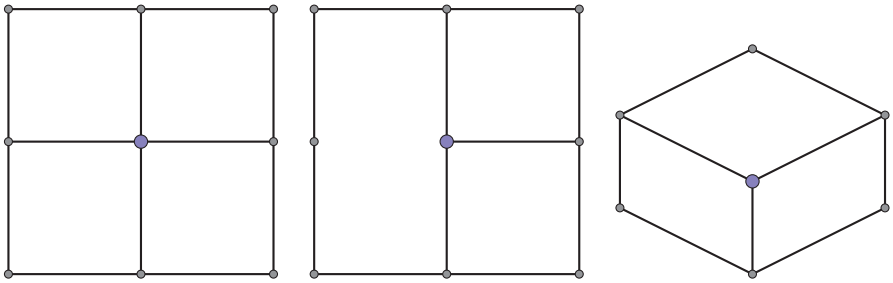


Figure III.2: Three different types of render loci. From left to right: regular, T, and star render loci.

- a T-point is defined as a grid (parameter) line ending in an orthogonal grid line, and
- a *Star*-point is defined as a point where several grid lines meet in a non-orthogonal way.

The render patch concept is closely connected to the description of the patch primitives of the GPU hardware. This provides some basic properties shared by all render patches.

1. The domain of a render patch is a parametric domain.
2. The parameter variables take values in the range $[0, 1]$.
3. In order to meet the requirements associated with the blending spline Hermite interpolation properties, the outer tessellation levels of two adjacent patch primitives must match.

Below follows a set of definitions that describe the concepts behind the names introduced above.

Definition III.4.1. A *render locus* is an extension to the points defined in [Lak13]. A *render locus* is defined as a locus in a render lattice graph and associated with a spline knot on a spline net. A render locus can be one of three basic types, depending on the point type of the spline knot, as described in [Lak13]. The three types are regular, T, and star.

Definition III.4.2. A *render patch* is an extension to the patch-type primitive of modern tessellation-based GPU architecture. (It names both the topological face and the rendered geometry.) The parametric domain of the render patch is limited by the boundary given by two, three, or four render loci. The number of render loci are decided by the type of the patch-type primitive, which for line, triangle, or quadratic patch-type primitives are two, three, or four, respectively. A point in the domain of a line, triangle, or quadratic render locus has a normalized position ((u), (u, v, w), (u, v), respectively) in the parametric domain $\Omega \in [0, 1]$.

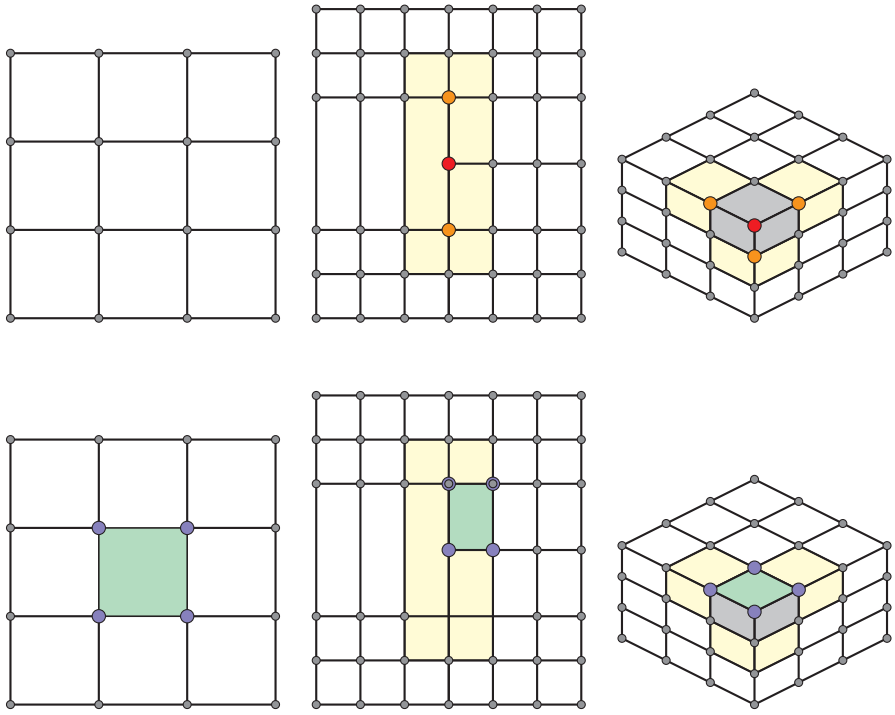


Figure III.3: Three different render lattices. A render patch is highlighted in each render lattice on the bottom row. The left column shows regular render loci. Regular and T-type render loci are shown in the middle column, whereas the right column contains regular and star-type render loci. The illustrations are provided to show the equivalence to the illustrations in [Lak13, Figure 4].

Definition III.4.3. A *render lattice* defines a graph topology on the spline knot nets of the blending spline construction. In a valid render lattice, the render loci are divided and partitioned such that the render lattice consists of adjacently connected render patches.

Examples of regular and irregular render lattices with quadratic render patches are shown in Figure III.3. Throughout this paper, we shall consider regular render lattices with quadratic render patches and regular render loci. T- and star-type render loci are subjects for future work.

III.4.1 Quadratic render patch with regular render loci

A render patch on a regular render lattice where all render loci are regular is defined in the following way

$$\begin{aligned}\hat{S}(u,v) &= \sum_{i=1}^{2}\sum_{i=1}^{2} \bar{L}_{i,j} \circ \omega_{i,j}(u,v)\, \mathfrak{B}_j(v)\, \mathfrak{B}_i(u) \\ &= \sum_{i=1}^{2}\sum_{i=1}^{2} \hat{L}_{i,j}(u,v)\, \mathfrak{B}_j(v)\, \mathfrak{B}_i(u)\ ,\end{aligned}$$

where $u, v \in [0,1]$ are parameters of the render patch, and $\omega_{i,j}(u,v)$ are "map-to-local" functions mapping the parametric domain of the render patch to the domain of the local surface associated with the given render locus.

The parametric domain of the patch-type primitive is normalized, and the knot interval is always local; therefore, the "map-to-local" functions can be simplified from how they are described in [Lak07]. For each parametric direction, u, v, the local mapping function, shown here for s, is defined as

$$\omega(s) = \gamma + \kappa \times s,$$

where γ is the parametric offset and κ is the scaling factor to the parametric domain, both with respect to the local surface.

Figure III.4 shows an example of a blending spline made up of 3×3 render loci, associated with one local surface each, and rendered over a render lattice consisting of four render patches.

III.4.2 Implementation strategies

The method proposed above provides a rendering strategy for "global" geometric objects, depending on the evaluation of "local" geometry, possibly in several layers. With respect to implementation, several topics could be considered related to optimization and performance, including the following

1. evaluation of blending functions,
2. evaluation of local geometry,
3. data organization, and
4. shader roles.

In this paper, we are not focusing on implementation details; however, we find it appropriate to comment on some of the issues mentioned above. Efficient GPU-centric evaluation of the original ERB was one of the essential topics. After it was addressed in [Zan14], it sparked the idea behind this paper.

The second and third topics are tightly coupled. For this reason we prefer to see them in connection. We propose two valid but different strategies here.

Figure III.4: The "Tower" surface: a blending spline tensor-product surface made up of 3×3 knots with associated local plane surface patches. The local surfaces in the corners are locally rotated in the xy- and xz-planes. The rendering lattice, in this case, is made up of 3×3 render loci (one for each spline knot locus) and four render patches. On the left is a shaded version of the resulting surface, while on the right is a wireframe version showing the tessellation of the four render patches.

The first is based on using a general evaluation scheme of pre-sampled data. It is unnecessary to customize the shaders for each type of local surface, such as, for example, a cap of a sphere or a Bézier patch, as each local surface has to be pre-evaluated. The primary issue with this is that the pre-evaluated data must be stored and managed; another is that the precision of the local surface data is given by the resolution of the pre-sampling approximation step. On the positive side, we note that, as a consequence of pre-evaluation, the evaluation time of a local patch would not depend on the patch type. Additionally, since the nature of the blending spline smoothly blends the local geometry, the entropy of the local surface evaluations can be lower.

The second strategy is using custom-built shaders for a given configuration of local render loci of a render patch. We mention the following drawback of this approach; on-the-fly evaluation of local surfaces could be more expensive than the look-up in a pre-evaluated evaluation buffer. Furthermore, each shader of a render patch must be regenerated whenever there is a change in the configuration of its render loci. Some significant upsides with this approach are that it facilitates code modularization and the possibility of pixel-accurate [Hje14; YBP14] rendering.

Code modularization is specific to individual GPU architectures and APIs. However, as an example, using OpenGL, one could divide each of the tessellation evaluation shader parts (BS evaluator, B-function evaluator, local patch evaluator, and others) into shader objects and choose the appropriate ones when used.

The last topic deals with achieving efficiency by shader design. Using the features provided by the tessellation shader, it is possible to generate patch geometry on the fly by circumventing the vertex shader altogether, providing only the coefficients of the local geometry to the shader. The tessellation shader generates the final geometry, and the local geometry only exists as an evaluation result. A change in position, orientation, or coefficient data of the local geometry would directly cause deformation to the rendered geometry. This change would not cause any additional computational costs as far as shader evaluation is concerned, and a stable framerate would be maintained. This literally means keeping the spline representation all the way to the graphics hardware. Furthermore, it facilitates affine spatial transformations of the spline coefficients before they are provided to the rendering pipeline.

III.5 Concluding remarks

We have introduced a method for smooth rendering of blending splines where standard features of the tessellation shader architecture are exploited. In addition, fundamental render patch types and their terminology have been proposed and described as well as strategies for implementation.

Two notable features of this method include

- predictable and constant rendering times, with respect to modification of the spline coefficients and local geometry, and
- the rendering method is local with respect to the render patch.

The method could be suitable for visualization in computer games or computer-generated imagery (CGI). Furthermore, since modification of the underlying spline construction adds little strain to the rendering method, it supports animation, simulations, and interactivity with little computational overhead.

The rendering method preserves the geometry description and topology until it is discretized in the hardware. Independent of implementation, the sampling is performed by the hardware instead of a defined procedure followed by pushing to the GPU.

Variable levels of detail (LODs) per render patch can be achieved by using well known methods for setting the inner tessellation levels through the control step of the tessellation shader [Hje14; YBP14]. The outer tessellation levels can be used to adjust and minimize artifacts over the boundary between two adjacent render patches. This is of interest when the blending spline construction is used in application areas resulting in large render patches, such as terrain representation.

The fact that the rendering method is strictly local (limited to the cover of the local surfaces associated with a render patch) and operates directly on the underlying spline construction makes applications within interactive geometric modeling and sculpting interesting topics for future work.

A preliminary implementation supporting rendering of regular tensor-product surface render latices was created, and the surface shown in Figure III.4 was rendered using this prototype, which provides a proof of concept. We propose focusing on efficient design for render patches containing T- and star-type render loci on irregular topology in the next stages of research and development.

Inter-operable features between GPGPU specialized architecture APIs, such as open computing language (OpenCL) [HM14; Mun14] or compute unified device architecture (CUDA) [Nvi14], and graphics architecture APIs, such as OpenGL [KHR2-; SA10] or DirectX [Mic5-], are available. For this reason, developing appropriate data structures and strategies for efficient data sharing and communication is desirable.

A more comprehensible study on the solution's efficiency should be conducted as part of any specific implementation.

References

[BDB15] Bratlie, J., Dalmo, R., and Bang, B. "Evaluation of smooth spline blending surfaces using GPU". In: *Curves and surfaces. 8th International Conference*. Ed. by Boissonnat, J.-D., Cohen, A., Gibaru, O., Gout, C., Lyche, T., Mazure, M.-L., and Schumaker, L. L. Vol. 9213. Lecture Notes in Computer Science. Springer, 2015, pp. 60–69 (cit. on p. 91).

[DBL09] Dechevsky, L. T., Bang, B., and Lakså, A. "Generalized Expo-Rational B-Splines". In: *International Journal of Pure and Applied Mathematics* vol. 57, no. 6 (2009), pp. 833–872 (cit. on p. 91).

[DLB06] Dechevsky, L. T., Lakså, A., and Bang, B. "Expo-Rational B-Splines". In: *International Journal of Pure and Applied Mathematics* vol. 27, no. 3 (2006), pp. 319–362 (cit. on p. 91).

[Hje14] Hjelmervik, J. "Direct Pixel-Accurate Rendering of Smooth Surfaces". English. In: *Mathematical Methods for Curves and Surfaces. 8th International Conference*. Ed. by Floater, M., Lyche, T., Mazure, M.-L., Mørken, K., and Schumaker, L. L. Vol. 8177. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, pp. 238–247 (cit. on pp. 97, 98).

[HM14] Howes, L. and Munshi, A. *The OpenCL API Specification (Version 2.0)*. 2014. URL: http://khronos.org/opencl (cit. on p. 99).

[KHR2-] KHRONOS group. *OpenGL*. 1992-. URL: http://khronos.org/opengl (cit. on p. 99).

[Lak07] Lakså, A. "Basic properties of Expo-Rational B-splines and practical use in Computer Aided Geometric Design". (Dr.philos.) PhD thesis. University of Oslo, 2007 (cit. on pp. 91, 96).

[LBD05] Lakså, A., Bang, B., and Dechevsky, L. T. "Exploring Expo-Rational B-splines for Curves and Surfaces". In: *Mathematical methods for Curves and Surfaces*. Ed. by Dæhlen, M., Mørken, K., and Schumaker, L. L. Nashboro Press, 2005, pp. 253–262 (cit. on p. 91).

[Lak13] Lakså, A. "ERBS-surface construction on irregular grids". In: *39th International conference applications of mathematics in engineering and economics AMEE13*. Ed. by Pasheva, V. and Venkov, G. Vol. 1570. AIP Conference Proceedings. AIP Publishing, 2013, pp. 113–120 (cit. on pp. 92–95).

[Mic09] Microsoft® corporation. *Direct3D 11 Features*. 2009. URL: http://docs.microsoft.com/en-us/windows/win32/direct3d11/direct3d-11-features (cit. on p. 91).

[Mic5-] Microsoft® corporation. *DirectX*. 1995-. URL: http://msdn.microsoft.com/directx (cit. on p. 99).

[Mun14] Munshi, A. *The OpenCL C Language Specification (Version 2.0)*. 2014. URL: http://khronos.org/opencl (cit. on p. 99).

[Nvi14] Nvidia. *CUDA C Programming guide v.6.5*. 2014. URL: http://developer.nvidia.com/cuda-toolkit-65 (cit. on p. 99).

[Sch+14] Schäfer, H., Nießner, M., Keinert, B., Stamminger, M., and Loop, C. "State of the Art Report on Real-time Rendering with Hardware Tessellation". In: *Eurographics 2014 - State of the Art Reports* (2014), pp. 93–117 (cit. on p. 91).

[SA10] Segal, M. and Akeley, K. *The OpenGL Graphics System: A Specification (Version 4.0 (Core Profile))*. Mar. 2010. URL: http://khronos.org/registry/OpenGL/specs/gl/glspec40.core.pdf (cit. on pp. 91, 99).

[YBP14] Yeo, Y., Bhandare, S., and Peters, J. "Efficient Pixel-accurate Rendering of Animated Curved Surfaces". English. In: *Mathematical Methods for Curves and Surfaces. 8th International Conference*. Ed. by Floater, M., Lyche, T., Mazure, M.-L., Mørken, K., and Schumaker, L. L. Vol. 8177. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, pp. 491–509 (cit. on pp. 97, 98).

[Zan14] Zanaty, P. "Application of Generalized Expo-Rational B-splines in Computer Aided Design and Analysis". PhD thesis. University of Oslo, 2014 (cit. on pp. 92, 96).

Paper IV

Exploring future C++ features within a geometric modeling context

Jostein Bratlie, Rune Dalmo

[**PREPRINT**] Bratlie, J. and Dalmo, R. "Exploring future C++ features within a geometric modeling context". In: *NIK: Norsk Informatikkonferanse*. Vol. 2019. 2019

Abstract

The development of the C++ programming language and its standard library has undergone a renaissance since the emerge of the C++11 standard. Through modern features, expansions to the standard library, and simplifications, the language has become more relevant than ever before. Comparing past and future feature sets (C++17, C++20, ...) are somewhat similar to comparing different programming languages. In this paper, we address how new and upcoming language features can ease the development of domain-specific application areas through features such as *non-intrusive inheritance*, *semantic compile-time polymorphism*, and type safety. We provide representative examples by application to differential geometry by modeling a hierarchical structure for parametric object evaluation.

IV.1 Introduction

Differential geometry is a field within mathematics where the theory of differential manifolds [Car76], such as curves and surfaces, and Riemannian spaces [Car92] are central. The R&D group Simulations at UiT Narvik has conducted research within this field since the mid-1990s, focusing on modeling techniques of spatial objects and spline theory. Realized implementations of these mathematical and geometric concepts and constructions are considered essential tools for visual understanding and verification throughout the R&D work process. As a result, an in-house software library named GMlib [LBK06], which aids in this aspect of the R&D work has emerged over the years. GMlib is a reasonably clean C++ library with only a few external dependencies besides the standard template library (STL). However, due to its age and maturity, the code base, written

in C++98, consists of major parts of legacy code, which now faces a set of challenges in the future. After an internal review in 2017, it was decided to construct a new prototype where the focus should be on statically descriptive tools for the differential geometric modeling aspects of our R&D work. In addition to historical reasons for continuing to use C++, we are fond of its closeness to hardware, optimization potential, no-overhead principles, and the support for generating bindings to other programming languages, such as, for example, Python. The central focus points for the development process are the following

- a light-weight header-only library, which follows the C++ core language guidelines [Iso21] and targets the latest C++ standard (current: C++17 [ISO17]),
- restrict library maintenance to key R&D activities, and
- optimize for rapid prototyping of domain-specific features and strong types.

After an analysis, we decided that the dependency needs of the new codebase were limited to three C++ libraries, besides the STL; namely, Blaze [Igl+12; Igl20] for basic linear algebra, OpenMesh [Bot+02] for triangular- and polygonal meshing and topology, and Qt [The19] for matters related to graphics, application framework, and demo control. The libraries were selected due to their use of modern C++ features, which we depend on, their proven stability, and their ability to target various architectures, from micro-controllers to smartphones and desktop- and backend systems. Both the old and new prototype libraries and demo applications can be found at the Nordic e-Infrastructure Collaboration (NeIC) software hub [UiT21], under the project "gmlib".

IV.2 Problem setting

The intended use of the new software library is as a tool to verify, very strictly, abstract mathematical concepts and constructions, where compile-time rather than runtime rules are utilized to catch constructional errors. A way to realize this is by using strongly typed expressions. In C++, this translates into a subset of features that enhances the static type system.

This paper focuses on specific new and future language features and how they can be utilized to enforce our R&D work. The paper introduces a set of design techniques and discusses how they are beneficial to the programming of geometric modeling concepts. The techniques are exemplified via building a small API throughout the paper. This API is used to model embedded spatial-hierarchical parameterized objects. Examples of such constructions can be

- a sphere, torus or Bézier surface, which all are two-dimensional objects that can be embedded in a three-dimensional space,
- or a multi-hierarchical object chain, such as the ones illustrated in Figures IV.1 and IV.2,

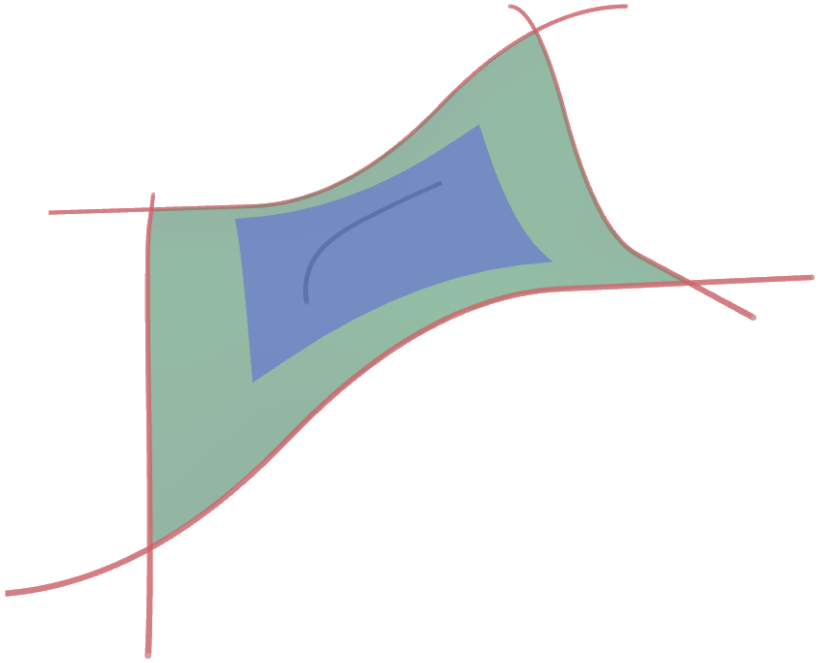


Figure IV.1: Left: a rendering of inheritance-hierarchical parametric objects; On the boundary, we have b-spline curves ($\mathbb{R} \mapsto \mathbb{R}^3$), from these we have sub-curves ($\mathbb{R} \mapsto \mathbb{R}$) which again are input to a bi-linear Coons patch surface, ($\mathbb{R}^2 \mapsto \mathbb{R}^3$). From this Coons patch, we have a sub-surface, plane ($\mathbb{R}^2 \mapsto \mathbb{R}^2$), and finally, we have a sub-curve, 5^{th}-degree Hermite curve ($\mathbb{R} \mapsto \mathbb{R}^2 \mapsto \mathbb{R}^3$).

of which the latter example is illustrated in Figure IV.1. Such problems, where the concatenated and multi-hierarchical are relevant scenarios, quickly become complex and challenging to model programmatically with confidence.

Implementation wise, if considering these objects as part of an infinite set, the natural modeling technique used in C++ is by utilization of abstract interfaces and virtual functions. This, in turn, leads techniques that build on dynamic casting and runtime type information (RTTI). On the other hand, by considering these hierarchical objects as parts of a finite set of objects, we can use a different set of C++ modeling techniques and utilize much more of the static type system, which again alleviates the use of runtime-centric techniques.

The paper is divided into three sections; *principal design techniques*, and an exemplified *parametric object API*, and a section which discusses pros and cons of these techniques and explores new and future language features. The considered new and future language features primarily enhance these principal design techniques and alleviates boilerplate code, reducing the complexity and

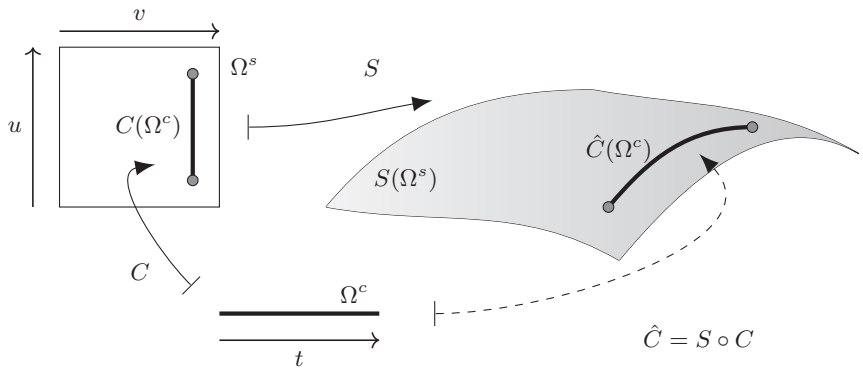


Figure IV.2: A mathematical sketch of the mapping $\hat{C} = S \circ C$

amount of programmatic errors.

IV.3 Principal design techniques

C++ is a mature well-behaving language possessing a set of features that are desirable for system design. One of these features is its well-defined type system, consisting of a finite set of built-in types and a single user-defined type: `class` /`struct`. The user-defined type is used to describe how objects are laid out in memory. Its associated member functions describe an object's semantics and related operations. As the language only consists of a single user-defined type, the inheritance model is equally well defined, and sub-objects with altered semantics are handled by the use of virtual tables. Furthermore, this yields a memory model that reflects the system memory for a target architecture and the requirement that all types are known at the point of compilation.

Due to its memory model, C++ supports casting between dynamic objects via runtime mechanisms. Dynamic casting is a powerful feature, but it comes with a possible high cost with respect to resources. The complexity of this and related mechanisms depend on the relationships between the dynamic objects and the hierarchical class structure. Additionally, that introduces a weak object model since an object's inheritance from a base type to a deduced type is subject to runtime checks. Nevertheless, this is the de facto method used to design object-oriented problems, where the possible objects from a given class hierarchy represent an infinite set where there is a parent-child relationship.

We propose to take an alternative approach to such design problems where we consider our infinite set of potential sub-classes to be finite at compile time. This allows us to use other mechanisms besides virtual tables and, for this purpose, disable the RTTI. In the following sections, this is exemplified through a small proposed API. We start by discussing a set of key C++ implementation techniques used to realize the static flavor of polymorphism, namely,

- non-intrusive inheritance,
- semantic compile-time polymorphism, and
- aggregated properties and transitive constructors.

IV.3.1 Non-intrusive inheritance

The term non-intrusive inheritance refers to an inheritance model where a derived class can inherit a realized base class, possibly unknown at the time of design, where the realized base class adheres to a set of static rules such that the memory model is preserved at the time of compilation. For instance, the derived class is defined in an external library, but the library's end-user can design the realized base class To facilitate non-intrusive inheritance one can utilize variadic class templates; `template <typename... Ts>`, which accepts one or more template parameters into a template pack. The ellipsis operator, ..., is then used to unpack the template arguments

```
struct SceneObject {}; struct Base {};

template <typename... Base_Ts> struct Object : Base_Ts... {};

using NonInheritingObject = Object<>;
using ObjectOfBase        = Object<Base>;
using SceneObjectOfBase   = Object<SceneObject,Base>;
```

IV.3.2 Semantic compile-time polymorphism

The standard mechanisms used in C++ to implement polymorphism are function overloading and virtual functions. The latter enables runtime-dynamic types where the semantics of a derived object can be reimplemented for a given virtual function in a derived class and then be called for the derived object dynamically through a pointer of the base class type. By flattening the structure, using a middle layer *kernel* type, utilizing *non-intrusive inheritance*, and exploiting function overloading, we can mimic much of the same behavior for semantic compile-time polymorphism. Consider the following type definitions

```
                    struct Base {};
<typename Kernel_T> struct Object : Kernel_T {};

template <Base_T> struct KernelA : Base_T {
  auto evaluate( int par ) { return Vector{par}; }
  auto evaluate() { return double(0.5); } };

template <Base_T> struct KernelB : Base_T {
  auto evaluate() { return double(0.5); } };

template <typename Obj_T, typename... Params_Ts>
auto evaluate(Obj_T t, Param_Ts... params){
  return t.evaluate(params...); }
```

The `Object` class inherits a potential kernel which again inherits a common base

```
using ObjectA = Object<KernelA<Base>>;
using ObjectB = Object<KernelB<Base>>;
```

The defined objects `ObjectA` and `ObjectB` share the same polymorphic API, where the return type is deduced by binding to the unconstrained placeholder type `auto`. Both object types can then be utilized in polymorphic contexts as follows

```
auto A = ObjectA; auto B = ObjectB;
auto a = evaluate(A,4); // Ok
auto b = evaluate(A);   // Ok
int  c = evaluate(A,4); // Compile error:    return type mismatch
auto d = evaluate(B,4); // Compile error: parameter type mismatch
```

IV.3.3 Aggregated properties

Static properties can be utilized to hold compile-time types or sizes, such as an object's spatial dimension or base unit type. These are properties that are changed, for instance, to increase computational precision (e.g., *long double* or *integer* over *floats*) or to specify the dimension of an embedding space (e.g., two-dimensional, $\mathbb{R}^2$, instead of three-dimensional, $\mathbb{R}^3$). Such internal properties can be defined using static constant expressions or `using` type definitions as part of *non-intrusive inheritance* (is-a relationship) or as internal aggregated type definitions (is-a or has-a relationship). Static properties are not part of an instantiated object by definition but can be stored if needed. As such, this mechanism is powerful and inflicts no runtime penalty. However, such properties are not aggregated through inheritance and must therefore be explicitly defined as follows

```
struct StaticProperties {
  using                 Unit      = int;
  static constexpr auto Dimension = 3ul; };

template <typename Props_T> struct Object {
  using                 Unit      = typename Props_T::Unit;
  static constexpr auto Dimension = Props_T::Dimension; };
```

IV.3.4 Transitive constructor inheritance

When utilizing a *non-intrusive inheritance model*, we need a mechanism to transitively aggregate the future constructors of a class type we do not yet know exists. This can be achieved using a combination of *templated constructors*, *variadic templates* and *perfect forwarding*. Let us exemplify it through an internal sub-object construction with a has-a relationship as follows

```
template <typename SubObj_T> struct Object {
  SubObj_T  sub;
  template <typename... Ts> Object(Ts&&... ts)
    : sub(std::forward<Ts>(ts)...){} };
```

Given an object type `A` for an internal sub-object with the following constructors

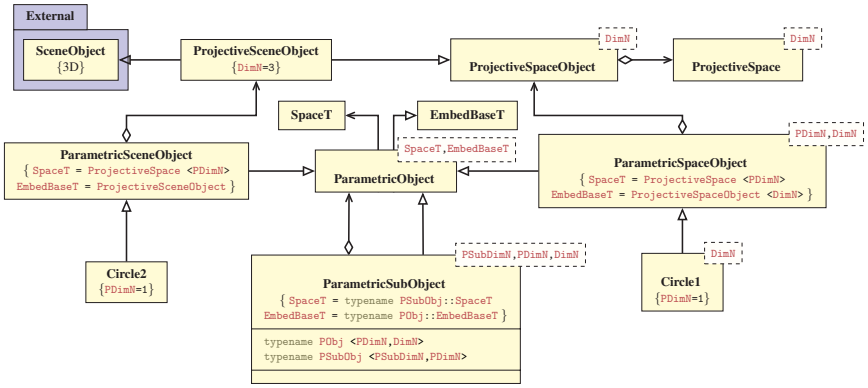


Figure IV.3: A templated model showing inheritance and semantic polymorphism.

```
struct A {
  explicit A(float f) {}
  explicit A(std::array<int,3> i) {} };
```

yields the following natural, but implicit aggregate, syntax

```
auto obj_one Object<A>(2.3f);
auto obj_two Object<A>({1,2,3});
```

IV.3.5 Simplifying API through type deduction

Finally, it is helpful to simplify an API without introducing new types. This can be achieved by using type definition via the `using` directive. Contrary to the pre-C++11 standard's `typedef` directive, `using` directives can be templated. Consider this API for a parametric object

```
template <size_t PSpaceDim, size_t EmbedDim> struct Object;
```

Specialized semantic APIss for curves and surfaces can then be constructed as follows

```
template <size_t EmbedDim> using Curve   = Object<1ul,EmbedDim>;
template <size_t EmbedDim> using Surface = Object<2ul,EmbedDim>;
```

IV.4 The parametric object API

Utilizing the above techniques, this section introduces a representative example API for the purpose of building parametric object spaces of the kind illustrated in Figures IV.1 and IV.2. A simplified API type diagram is shown in Figure IV.3.

IV.4.1 Projective space

The projective space is common when working with visual parametric modeling. It is an affine mathematical space (see literature on differential- or Riemannian geometry [Car76; Car92]), consisting of points and vectors with respect to a perspective projection such that parallel lines end up in a point at infinity, i.e., the horizon. The projective space meta-data description consists of its affine space containers for points and vectors in addition to a numerical unit data type and a space dimension. The space is usually represented as a frame in the shape of a homogeneous matrix. We model the meta-information of this mathematical space statically by using an *information only structure* as follows

```
template <typename Unit_T, size_t Dim_T> struct ProjectiveSpace {
  static constexpr auto Dim = Dim_T;   // Space dimension
  using Unit    = Unit_T;              // Storage unit type
  using Point   = Vector<Unit, Dim>;   // Affine space data types
  using Vector  = Vector<Unit, Dim>;
  using Frame   = Matrix<Unit, Dim>; };
```

From this, one can model a projective space object using a static *has-a* relationship

```
template <typename EmbedSpace_T> struct ProjectiveSpaceObject {
  using EmbedSpace = EmbedSpace_T;        // Projective space
  using Unit = typename EmbedSpace::Unit; // Type aggregation
  /* ... */
  Frame frame = {/* identity */};         // Affine space data structure
  void translate( Vector ) { /*math*/ };  // Affine operations
  /* ... */ };
```

IV.4.2 Parametric objects

A parametric object is a user-defined type describing a differential mapping from an n-dimensional parameter space into a m-dimensional embed space, under certain restrictions. This can be modeled using the principles of *non-intrusive inheritance*, where a parametric object inherits a template kernel type, `Kernel_T`, realizing the actual parametric object as follows

```
template <typename Kernel_T>
struct ParametricObjectImpl : Kernel_T {
  // Context
  using Kernel = Kernel_T;
  // Embed Space
  using EmbedSpaceObj = typename Kernel::EmbedSpaceObj;
  using EmbedSpace    = typename Kernel::EmbedSpace;
  using Unit          = typename Kernel::Unit;
  /* ... */
  // Parametric space
  using PSpace        = typename Kernel::PSpace;
  using PSpaceUnit    = typename Kernel::PSpaceUnit;
  /* ... */
```

The specific space types are inherited from the kernel as aggregated properties, and the member functions are inherited from the kernel according to the rules

of inheritance. These can be used to construct chaining operations; such as transformations or evaluation with respect to a *parent* space (`frame`)

```
  auto evaluateParent(PSpacePoint par) {
    return frame * evaluate(par); }
  /* ... */
```

The kernel's constructor is inherited using *transitive constructor inheritance*, effectively translating the kernel's constructor into the parametric object's own constructor. This can also be done for other *internal* kernel methods, such as private sub-methods.

```
  template <typename... Ts>
  explicit ParametricObjectImpl(Ts&&... ts)
    : Kernel(std::forward<Ts>(ts)...) { }
}; // END ParametricObjectImpl
```

IV.4.3 A parametric object kernel

A specialized parametric object is then realized as a parametric object of a provided kernel. As an example, the parametric object kernel, `CircleKernel`, is constructed as a user-defined type inheriting the projective space object as a template type `EmbedSpaceObj_T`. Then as with the parametric object itself, we use *aggregated properties* to inherit the static types. Additionally, we define equivalent projective space types for the parameter space itself and required utility types, prefixed `PSpace`. This is exemplified using a parameterized circle $C(t) = (x(t), y(t), z(t)) : \mathbb{R} \mapsto \mathbb{R}^3$.

```
template <typename  EmbedSpaceObj_T> struct CircleKernel
  : EmbedSpaceObj_T {
  using EmbedSpaceObj= EmbedSpaceObj_T;                  // Embed Space
  using EmbedSpace    = typename EmbedSpaceObj::EmbedSpace;
  /* ... */
  using PSpace        = space::ProjectiveSpace<1ul,Unit>; // PSpace
  using PSpaceUnit    = typename PSpace::Unit;
  static constexpr auto PSpaceDim = PSpace::Dim;
  /* ... */
  using PSpaceBoolArray = array<bool, PSpaceDim>;        // Utility
  using PSpaceSizeArray = array<size_t, PSpaceDim>;
```

The above code in fact represents boilerplate code for any parametric curve. The kernel-specifics for the parametric circle, such as data members, constructor and semantic polymorphic functions, can then be defined as follows

```
  Unit r;                                       // Member: radius
  template <typename... Ts>                     // Transitive constructor
  CircleKernel(Unit radius = 3, Ts&&... ts)
    : Base(std::forward<Ts>(ts)...), r{radius} {}
                                                // Semantic polymorphic
                                                    functions
  auto            evaluate(PSpacePoint par) const {
     const auto& [t] = par;

     const auto x    = r * std::cos(t);
     const auto y    = r * std::sin(t);
```

```
11
12      if      constexpr (Dim == 2) return Point{x, y};    // 2D
13      else if constexpr (Dim == 3) return Point{x, y, 0}; // 3D
14    }
15    PSpaceBoolArray isClosed()       const { return {true}; }
16    PSpacePoint     startParameter() const { return {0}; }
17    PSpacePoint     endParameter()   const { return {2*M_PI}; }
18  }; // END CircleKernel
```

The template dependent *constexpr if* statement enables us to specify different return types for each supported embed space dimension.

IV.4.4 Sub-space objects

The driving motivation to develop such an API has been to enable strong typing of hierarchical object structures such as the ones addressed in Section IV.2. Contrary to the principal parametric object, a sub-object kernel is defined by a parametric object embedded in the parameter space of another parametric object. This can be achieved by defining the kernel and implementation of the sub-object slightly different from the parametric object type itself. However, as an invariant, the internal type-definitions should be kept identical to the ones of the parametric object, such that sub-object chains can be defined.

The kernel is defined from three types; the subbed parametric object, `ParametricObject_T`, a parametric object embedded in that parametric object's parametric space, `PSpaceObject_T`, and an embed space object, `EmbedSpaceObject_T`, which usually is the same as the embed space of the sub-object. This gives us a parametric object with three distinct spaces: the embed space of the parametric object, `EmbedSpace`, the parametric space of the sub-object, `PSpace`, and the parametric space of the parametric object, `ParametricObject_PSpace`.

```
1  template <typename PSpaceObject_T, typename ParametricObject_T,
2            typename EmbedSpaceObject_T>
3  struct CurveInSurfaceKernel : EmbedSpaceObject_T {
4    using PSpaceObject            = PSpaceObject_T;      // Context
5    using ParametricObject        = ParametricObject_T;
6    using EmbedSpaceObject        = EmbedSpaceObject_T;
7
8    using EmbedSpace              = typename EmbedSpaceObject::EmbedSpace;
9    using PSpace                  = typename PSpaceObject::PSpace;
10   using ParametricObject_PSpace = typename ParametricObject::PSpace;
11
12   using Unit                    = typename EmbedSpace::Unit;
13   /* ... */
```

This leaves us with a similar boilerplate code as for the parametric circle kernel but with an additional typeset representing the common parametric space. Continuing, the sub-object data members, constructor, and polymorphic parametric object methods are defined as follows.

```
1    PSpaceObject      pspace_object;      // Members
2    ParametricObject* parametric_object;
3
4    template <typename... Ts>             // Transitive constructor
```

```
  explicit SubCurveKernel(ParametricObject* obj, Ts&&... ts)
  : Base(), pspace_object(std::forward<Ts>(ts)...),
                          parametric_object{obj} {} };

  auto evaluate(PSpacePoint par) const  // Semantic polymorphic functions
  {
    const auto pspace_res = pspace_object.evaluate(par);
    const auto parametric_object_res =
      parametric_object->evaluate(pspace_res);
    return parametric_object_res;
  }
  auto isClosed() const { return pspace_object.isClosed(); }
  /* ... */
}; // END CurveInSurfaceKernel
```

IV.4.4.1 Parametric sub-object implementation type

The implementation type for the parametric sub-object type is a slight variation of the parametric object's type. Types are aggregated from the sub-object-in-object kernel, which transitively calls the right constructor combination. Again, it must retain the same type definition interface as the `ParametricObjectImpl` such that it also can be used in sub-object chain constructions.

```
template <typename Kernel_T> struct ParametricSubObjectImpl
  : ParametricObjectImpl<Kernel_T> {
  // Context
  using Kernel           = Kernel_T;
  using PSpaceObject     = typename Kernel::PSpaceObject;
  using ParametricObject = typename Kernel::ParametricObject;
  using EmbedSpaceObject = typename Kernel::EmbedSpaceObject;
  // Aggregated types
  using PSpace     = typename Kernel::PSpace;
  using EmbedSpace = typename EmbedSpaceObject::EmbedSpace;
  using ParametricObject_PSpace =
    typename Kernel::ParametricObject_PSpace;
```

Object creation is handled through a transitive constructor, which can be enforced by deleting the sub-objects default constructor.

```
  ParametricSubObjectImpl() = delete; // Delete default constructor
  template <typename... Ts>        // Transitive Constructor
  explicit ParametricSubObjectImpl(ParametricObject* obj,
                                   Ts&&... ts)
    : Base(obj, std::forward<Ts>(ts)...) {}
}; // END ParametricSubObjectImpl
```

IV.4.5 User space API

C++ template code naturally results in low readability API's, however, as discussed in Section IV.3, this can be mended utilizing `using` type-definitions

```
// ParametricObject
template <template <typename> typename Kernel_T,
          typename EmbedSpaceObject_T>
using ParametricObject
```

```
  = ParametricObjectImpl<Kernel_T<EmbedSpaceObject_T>>;

// ParametricSubObject
template <template <typename> typename PSpaceKernel_T,
          template <typename, typename, typename>
          typename Kernel_T,
          typename ParametricObject_T>
using ParametricSubObject =
  ParametricSubObjectImpl<Kernel_T<
    ParametricObjectImpl<PSpaceKernel_T<ProjectiveSpaceObject<
      typename ParametricObject_T::PSpace>>>,
    ParametricObject_T,
    ProjectiveSpaceObject<
      typename ParametricObject_T::EmbedSpace>>>;
```

which leads to the following API usage

```
using ProjSpace    = ProjectiveSpace<double,3ul>;
using ProjSpaceObj = ProjectiveSpaceObject<ProjSpace>;
using Circle       = ParametricObject     <CircleKernel, ProjSpaceObj>;
using Torus        = ParametricObject     <TorusKernel,  ProjSpaceObj>;
using SubCircle    = ParametricSubObject  <CircleKernel, SubCurveKernel,
     Torus>;

auto circle          = Circle();
auto torus           = Torus();
auto torus_subcircle = SubCircle(&torus, 2.0);
```

IV.5 Concluding remarks

Utilizing the principles of *non-intrusive inheritance* and *semantic compile-time polymorphism*, we can redefine the object space of our modeling problem from an infinite to a finite set of possible objects. This allows us to design our C++ programs using static compile-time techniques, which, in turn, lets us, most notably *move viable logic from runtime to compile-time*, and apply *stronger types*. Consequently, fewer potential runtime errors occur since they can be caught at compile-time, and we can dismiss entire type categories by definition. Furthermore, the paradigm enables us to apply static analysis tools to problems where we earlier had to perform dynamic analysis. This again reduces errors and potentially leads to faster-runtime code, for instance, by dismissing branching opportunities.

On the other hand, template programming makes libraries more complex, which puts more responsibility on library programmers to construct understandable APIs. One drawback of a templated codebase is increased compile times and compile-time resources. If not handled with care, the increase in compile-time and resources can grow exponentially. However, this will be alleviated with the upcoming C++20 feature; *modules*. Modules will mitigate the compile-time overhead endured from repeated recursive header includes as well as the header inclusion mangling itself. Another drawback, if not handled with care, is the combinatoric nature of templates. As all function candidates, types and branches

must be known at compile-time, all permutations of possible template combinations are computed and evaluated by the compiler. This can then make both compile times, and the size of executables grow exponentially.

IV.6 Future work

Let us look at future language features, which will further help verify our R&D work and ease the development of such hierarchical class structures. C++20 introduced a new set of features. This is the most significant jump in a standardized feature level since the emerge of C++11. In this section, we argue what we see as the most prominent feature with respect to the current proposed API, namely *Concepts*. Furthermore, we briefly touch on a set of future features which are in development and planned for inclusion in standardized C++ over the next decade.

IV.6.1 Concepts

In today's codebase, *concepts* [SM19] are represented by *type-traits* and enforced through the SFINAE-mechanism (substitution failure is not an error). SFINAE, together with *type-traits*, limits the type space a template variable can occupy, and is used to remove template function candidates, at compile-time, if type deduction fails.

Concepts, on the other hand, builds on the placeholder type `auto`. The type `auto` is the value-equivalent of `typename`. It is deduced at compile-time, upon definition. With

```
1 auto i = int{4};
```

the type of the variable `i` is deduced to integer, while in the following example the type is deduced to the return-type value of the factory function.

```
1 auto curve = circleFactory(3.0);
```

A concept type is a restricted placeholder type. Imagine an algorithm that finds the intersection between two parametric objects. In today's codebase, a generic intersect algorithm is represented as a templated function over two unconstrained typenames; returning some result

```
1 template <typename ObjA_T, typename ObjB_T> auto intersect(ObjA_T, ObjB_T);
```

In theory, types `ObjA_T` and `ObjB_T` can take any type. The error checking is delayed to later in the code where the functionality and types are realized. This leads to obscure error messages often hidden inside a static template recursion layer.

An intersect method written using concept mechanisms would look something like the following (using the work group's short-hand notation [Sut15])

```
1 template <> IntersectResult auto intersect(Curve auto, Surface auto);
```

where `Curve`, `Surface` and `IntersectResult` are concept placeholder types. In this scenario, the compiler deduces the placeholder types at the time of binding and

can give an error message based on the restricted concept types, for instance: "function parameter one is not a curve, because ...".

The improved error messages are an appreciated side effect, but the real benefit is that we can model concepts by their unique and shared properties rather than restricting their incidental side effects.

IV.6.2 Reflection, metaclasses, and contracts

Reflection and metaclasses introduce compile-time functionality, building on the C++20 feature `consteval`, to either reflect upon existing user-defined source elements or inject program source fragments at *compile-time*. Using syntax rules from the proposal paper [Sut19], we can define a parametric circle type as follows

```
class(ParametricCurve) Circle {
  using Unit                = double;
  static constexpr auto Dim = 3ul;

  PSpacePoint     startParameter() const { /**/ };
  PSpacePoint     endParameter() const   { /**/ };
  PSpaceBoolArray isClosed() const       { /**/ }
  Point           evaluate(PSpacePoint par) const { /**/ } };
```

The feature then dictates that our user-defined type, `Circle`, needs to comply with the rules posted through the meta-class `ParametricCurve`. If so, a set of the code fragments, completing the parametric curve construction, will be statically generated. This does not inflict any implicit inheritance.

Contracts, on the other hand, are a runtime tooling feature that documents and enforces invariants. For instance, in the parametric object construction, we could apply the following value invariants (this example uses syntax from the proposal [Rei+18]).

```
Point evaluate(PSpacePoint t) const
  [[ expects:             t >= startParameters() and t <= endParameters() ]]
  [[ ensures default res: not std::isnan(res) ]]
{
  [[ assert: not std::isnan(t) ]]
}
```

The `expects` directive is applied to input parameters, the `ensures` directive can reflect constraints onto the return value, and the `assert` directive produces assertion conditions. Contracts provide functionality that enables us to turn on restrictions during development.

IV.7 Acknowledgments

The prototype library, GMlib2, is a work in progress; therefore, the paper is accompanied by a small source code base containing a proposed principal implementation example of the discussed API, included in appendix A. The authors acknowledge the equal contribution norm [Tsc+07] associated with the alphabetical listing of authors.

References

[Bot+02] Botsch, M., Steinberg, S., Bischoff, S., and Kobbelt, L. "OpenMesh: A Generic and Efficient Polygon Mesh Data Structure". In: *OpenSG Symposium 2002*. 2002 (cit. on p. 104).

[BD19] Bratlie, J. and Dalmo, R. "Exploring future C++ features within a geometric modeling context". In: *NIK: Norsk Informatikkonferanse*. Vol. 2019. 2019 (cit. on p. 103).

[Car76] Carmo, M. P. do. *Differential Geometry of Curves and Surfaces*. New Jersey: Prentice-Hall, 1976. URL: http://books.google.no/books?id=1v0YAQAAIAAJ (cit. on pp. 103, 110).

[Car92] Carmo, M. P. do. *Riemannian Geometry*. Mathematics (Boston, Mass.) Birkhäuser, 1992. URL: http://books.google.no/books?id=uXJQQgAACAAJ (cit. on pp. 103, 110).

[Igl+12] Iglberger, K., Hager, G., Treibig, J., and Rüde, U. "Expression Templates Revisited: A Performance Analysis of Current Methodologies". In: *SIAM Journal on Scientific Computing* vol. 34, no. 2 (2012), pp. C42–C69 (cit. on p. 104).

[Igl20] Iglberger, K. *Blaze C++ Linear Algebra Library*. (Bitbucket). Aug. 2020. URL: http://bitbucket.org/blaze-lib (cit. on p. 104).

[ISO17] ISO. *ISO/IEC 14882:2017 Information technology — Programming languages — C++*. Fifth. International Organization for Standardization, Dec. 2017, p. 1605 (cit. on p. 104).

[Iso21] IsoCpp. *C++ Core Guidelines*. Ed. by Stroustrup, B. and Sutter, H. (GitHub). May 2021. URL: http://github.com/isocpp/CppCoreGuidelines (cit. on p. 104).

[LBK06] Lakså, A., Bang, B., and Kristoffersen, A. R. *GM_lib, a C++ library for geometric modeling*. Narvik, Norway: Narvik University College, 2006 (cit. on p. 103).

[Rei+18] Reis, G. D., Garica, J. D., Lakos, J., Meredith, A., Myers, N., and Stroustrup, B. *Support for contract based programming in C++*. CEWG proposal P0542R5. ISO/IEC – JTC1/SC22/WG21, 2018 (cit. on p. 116).

[SM19] Stepanov, A. and McJones, P. *Elements of Programming*. Authors'/second. Semigroup Press, 2019 (cit. on p. 115).

[Sut19] Sutter, H. *Metaclass functions: Generative C++*. SG7/EWG proposal P0707R4. ISO/IEC – JTC1/SC22/WG21, 2019 (cit. on p. 116).

[Sut15] Sutton, A. *Programming languages — C++ Extensions for Concepts*. Proposal N4549. ISO/IEC – JTC1/SC22/WG21, 2015 (cit. on p. 115).

[The19] The Qt Company. *Qt - Complete software development framework*. 2019. URL: http://qt.io (cit. on p. 104).

[Tsc+07] Tscharntke, T., Hochberg, M. E., Rand, T. A., Resh, V. H., and Krauss, J. "Author sequence and credit for contributions in multiauthored publications". In: *PLoS biology* vol. 5, no. 1 (2007), e18 (cit. on p. 116).

[UiT21] UiT - The Arctic University of Norway. *GMlib/GMlib2 C++ Geometric modeling library*. (NeIC). May 2021. URL: http://source.coderefinery.org/gmlib (cit. on p. 104).

Paper V

Blending spline polygon surface over arbitrary poly-mesh topology

Jostein Bratlie, Rune Dalmo

[PREPRINT] Bratlie, J. and Dalmo, R. "Blending spline polygon surface over arbitrary poly-mesh topology". Under revision.

Abstract

Polygon meshes are common representations of geometrical objects in computer graphics and solid modeling. Continuous surface representations are required for many application areas. Several methods exist for spline-based surfaces of different brands over triangulations, quadrangulations, and mixed combinations of triangles and quadrilaterals. In this paper, we present a new method for representing a smooth surface in terms of blending splines over arbitrary poly-mesh topology. The new surface provides a flexible construction that facilitates user interactivity and editing. The main results are related to the applicability to meshes containing faces of convex and concave types and with different numbers of sides while preserving a smooth surface over the edges of the input mesh. In the process of derivation of this representation, we investigate some properties of the construction, focusing on the smoothness over the internal edges of its resulting patches.

V.1 Introduction

Freeform surface editing is a modeling technique used mainly in computer aided design (CAD) tools to model complex shaped surfaces, such as the skin of a 3D character in a virtual world, car bodies, and more. Most major CAD and 3D modeling tools, such as SolidWorks®, Autodesk®AutoCAD, and 3ds Max, Rhinoceros®, or Blender, implement one of the industry standards for the purpose of representing surfaces. Examples of such surface representations are non-uniform B-splines (NURBS), Coons patches, or T-splines, to name a few. These constructions have some commonalities; most notably, their coefficients are spatial points, and increasing the degree of freedom is usually performed via either degree elevation (under the penalty of moving a step away from local towards global support) or subdivision.

Blending splines belong to another branch of freeform surface representations, where local surfaces are blended together to form global surface patches.

These constructions can increase the degree of freedom through local geometry transformations while maintaining their minimal support properties. Blending splines are structurally more complex when compared to, for example, NURBS. However, characteristic features such as minimal support and local surface geometry coefficients make the construction flexible and therefore appealing for use in freeform editing and artistic sculpting.

In this work, we develop blending splines for arbitrary poly-mesh topology while seeking to maintain these desired features. This means moving away from the specialized methods for splines over triangular [LS07; SS06] or quadrangular [BZK09] meshes and their partitioning techniques [SW12] and rather trying to deal with the challenges of polygonal domain control.

V.2 Exposition

The paper is organized as follows. We start by giving a short introduction to the essential theoretical background, i.e., patchworks and blending splines, followed by an overview of related work. Then we describe the problem setting, the main challenge, and the construction on an overall level. Rather than diving into details in the high level description, the appropriate subsequent topical sections are forward-referred. After that, follows detailed topical sections describing the entirety of the construction. After the detailed exposition of the new construction, we present some representative examples and discuss the core properties. This constitutes the bulk of the paper. Finally, we give our concluding remarks and suggest some topics for future research.

V.3 Patchwork surface representations

See Section 2.1.

V.4 Blending surface representations

See Section 2.2.

V.5 Related work

In this section, we give an overview of previous related work in order to put the new method into context.

V.5.1 Spline spaces over quadrangular meshes

The tensor-product mesh structure is a well-known kind of spline surface construction over regular quadrangular meshes. Mesh structures containing irregular grid joints are sometimes required in practical applications. For example, the advent of IGA [CHB09] has triggered an interest in local refinement. Traditional tensor-product splines lack local refinement as a consequence of

the underlying regular grid topology. T-spline [Sed+03], LR-spline [DLP13], and THB-spline [GJS12] surface constructions break the tensor-product mesh structure by introducing local refinement, enabling irregular grid topology.

V.5.2 Polygonal surface patches

Multi-variate parametric surface patches (in more than two variables) have been an active area of research for decades. One natural approach is to define the parametric domain via some type of polygon. The two classical approaches within CAGD are courtesies of P. Bézier and P. de Casteljau, whose surfaces have a tensor product- and triangular domain, respectively [Far02]. Later, C. Loop and T. DeRose introduced S-patches [LD89], a class of surface patch representations that generalize and unify triangular- and tensor-product Bézier surfaces by allowing multi-sided domains. Recently, through a series of publications [SV18; VRS11; VSK16; VSK17], constructions that generalize the concepts of P. Bézier and P. de Casteljau in a simpler manner than the S-patch has been introduced. Most notable are the composite ribbon patches [VSK16; VSK17], which generalizes Coons patches, the generalized Bézier patches [VSK17], which constructs Bézier-type polygon patches over convex polygons, the generalized ribbon-based Bézier patches [SV18], which extends the generalized Bézier patches to concave polygonal domains, and Gregory generalized Bézier- and S-patches [HK18]. The parameterization of these general polygon-centric solutions relies on GBCs; see for example [Flo15] for an overview.

V.6 Construction overview

In this paper, we derive a polygon-based blending spline surface construction by extending the classic tensor-product. The new construction is developed to support surfaces over arbitrary poly-mesh topology.

This extension turns out to be non-trivial and can introduce some confusion since the proposed solution combines theory from different application areas. Therefore we separate parts of the construction into layers as illustrated in Figure V.1. The overall blending construction and geometry are represented by the *surface- and blending sub-layer*. These layers represent existing blending spline tensor-product constructions. The *reparameterization layer* represents techniques used to extend the tensor-product concept to an arbitrary poly-mesh construction. The *topology layer* deals with topological consistency across hierarchical layers and is directly associable with control mesh structures used for approximation purposes. This is equivalent to alternative manifold-based constructions.

The main technical sections of this paper are written such that the theory follows the structure familiar with classic blending spline constructions. Due to its hierarchical and layered structure, imposed by the required polygonal constructions, the presentation can be somewhat counter-intuitive. Therefore,

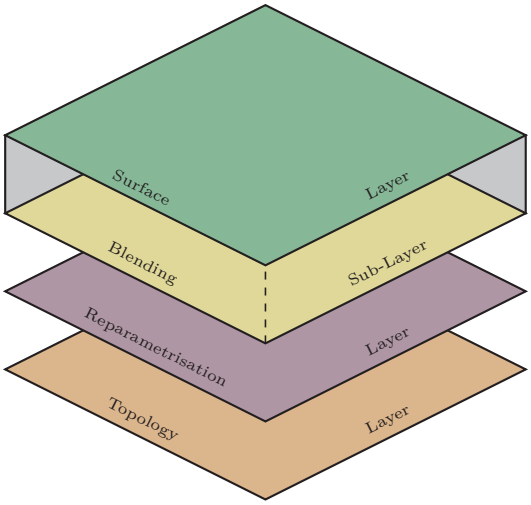


Figure V.1: The constructions logical layers; surface, blending, parametric reparameterization, and topology.

this section is provided to give a general overview of the construction while forward-referring subsequent sections.

V.6.1 Motivation, problem-setting, and contributions

The overall motivation for the current work is summarized in the following

1. construct a smooth blending spline surface over an arbitrary poly-mesh topology,
2. obtain a flexible, editable construction applicable to interactive modeling, and
3. facilitate parallel computation kernels for graphics computations as well as general-purpose computations, i.e., GPGPU.

Issue 1 was first addressed in [Lak07, p. 6.7] for surfaces over triangulations. A vital result was recognizing the criteria for determining a correct parameterization (constant parameter lines) while preserving acceptable visual performance (unwanted geometric artifacts due to parameter space distortion) and simple evaluation. A tensor product solution for different kinds of parameter line joints was proposed in [Lak13]. Consequently, the cover of the local geometry was increased under the penalty of losing the minimal support property near irregular joints.

Issue 2 refers to a sufficiently non-complex and concise construction that facilitates iso-geometric modeling and analysis. The base construction is sought

to be minimal disjoint with respect to hierarchical complexity, where the blending spline coefficients (local geometry) are preferred to be simple local constructions, e.g., Bézier-type surfaces. This is to facilitate intuitive interactive editing through de facto industry techniques.

Issue 3 was first addressed for tensor-product surfaces in [BDB15]. An implementation based on the current work can utilize the same implementation techniques; however, the implicit knot interval domains must be partitioned in sub-domains fitting the supported patch-type primitives.

V.6.1.1 Problem setting and contributions

Through the current work, we present a new surface construction based on blending splines over arbitrary polygon-mesh topology. The main problem-setting is to control implicit parametric polygon domains through explicit hierarchical construction. The new contributions can be summarized in the following;

- preservation of smooth parameterizations across patch boundaries by controlling overlapping parametric polygon domains, and;
- an implicit/explicit knot vector representation through the use of a knot graph.

V.6.2 Nomenclature and notation

We continue by describing notation and nomenclature such that we can, in a clean manner, relate domain partitions both across the various construction layers as well as in a single construction layer. To simplify the notation, for the reader, we minimize the use of indices in the notation. However, indices are necessary when we need to describe elements such as loop-controls, summations and ambiguous conditions. Some proficiency in the basic theory of tensor product- and triangular Bézier and B-spline surfaces is assumed. (See, e.g., [PBP02] or [Far02] for an introduction to those topics.) Additionally, contrary to a surface patchwork construction, which has a flat structure with respect to geometry, understanding the blending spline structure hierarchy can benefit from a relational component diagram. Figure V.2 provides a component-wise illustration of the construction with nomenclatural annotations and introduces a color scheme for its subsystems utilized in the sequel.

V.6.2.1 Topology control

A structure conformal with an extended half-edge graph structure [Ket99], $G = (Q, H, E, F, \Pi = (\Pi^\sigma, \Pi^\phi))$, is utilized. G is a tuple containing the set of vertices Q, directed half-edges H, edges E, and quad faces F. Additionally, for spline control, we equip the graph tuple G with an extension $\Pi = (\Pi^\sigma, \Pi^\phi)$ which contains a set of knot multiplicities Π^σ, and a set of parametric edge distances Π^ϕ. The individual elements are denoted by using corresponding lower case characters.

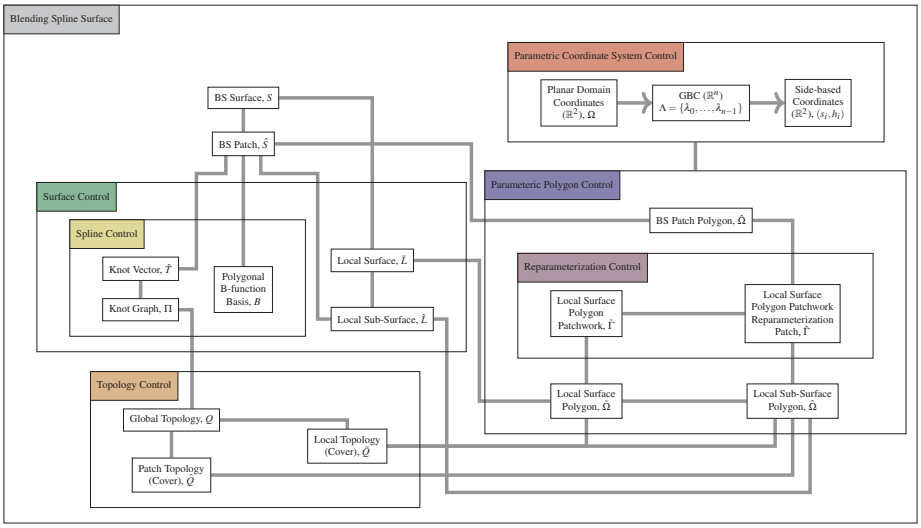


Figure V.2: Component-wise overview of the blending spline construction hierarchy. The individual subsystems and layers are indicated via introducing a color scheme.

V.6.2.2 Domain partitions

Bar-, hat- and no accents, e.g., $\bar{Q}$, $\hat{Q}$ and Q, denote domains associated with local surfaces and local surface covers, blending spline patches and patch covers, and global domains, respectively. More details are provided in Section V.7.

V.6.2.3 Geometry construction layers

The following symbols denote various geometry- and space layers, such as topology, blending spline patches, local surfaces, parameter space polygons and reparameterization patchworks

Q: control mesh, e.g., local control mesh, $\bar{Q}$,

S: global geometry, e.g., blending spline patch, $\hat{S}$,

L: local geometry, e.g., local sub-surface, $\bar{L}$,

Ω: parametric polygon, e.g., local surface parametric polygon, $\bar{\Omega}$, and

Γ: reparameterization, e.g., local reparameterization patchwork, $\bar{\Gamma}$.

V.6.2.4 Additional notation

Additionally, we have the following notation

Π : knot graph,

T : knot vector,

$\mathfrak{B}$: B-function,

Λ : GBC; $\Lambda = (\lambda_0, \ldots, \lambda_n)$, and

(s, h) : side-based coordinate.

The subscript notation is utilized to relate objects which belong to different domain partitions, e.g., $\bar{Q}_{\hat{q}}$ is a local sub-cover (being a patch cover associated with an adjacent local surface cover).

V.6.3 Topology notation vs. the half-edge structure

In the case of patchworks, it is common to use a mesh- or graph-centric notation, for instance, where a patch is related to a face on the mesh. However, our construction is hierarchical, and the meaning of the mesh notation depends on the context. As an example, a blending spline patch covering a knot interval is not equivalent to the local surface cover over a local surface domain. Specific notation for our polygon construction is described in detail in Section V.7.

V.6.4 Parameterization of the polygon construction

The global topology, or knot interval configurations, of triangular- and tensor-product blending splines are considered trivial due to parameterization compatibility across edges and precisely overlapping local geometry structures in the respective parametric domains (see Section V.4).

On the other hand, polygonal constructions introduce parameterizations with non-trivial domains for the overlapping parametric spaces of adjacent local sub-patches and their shared knot intervals. Additionally, the non-explicit evaluation of the GBC introduces an extra challenge to the overlapping parametric domains of the local surfaces and blending spline patches, which causes the intrinsic parametric lines to be discontinuous across the edges (between blending spline patches).

The concept of reparameterization is well known from differential geometry for parametric curves and surfaces. This reparameterization can be classified as an affine transformation in most applications since the resulting parameter space is isomorphic to the original, as illustrated in Figure V.3. Blending of two parametric tensor-product curves, surfaces, or volumes is well defined due to the affine properties, since such mapping between two overlapping parametric domains is bijective by nature.

In the case of multi-variate parametric surfaces, where the parametric domain is triangular or polygonal, a change of coordinate system is useful. The explicit parameter space is usually represented by generalized barycentric coordinate mappings [Flo15]; $\Omega \mapsto \Lambda,\ \Omega \subset \mathbb{R}^2,\ \Lambda \subset R^n$, where n is the number of sides in a polygon ($n = 3$ for triangles), and $\sum_1^n \lambda_i = 1$. (Triangles are a special case,

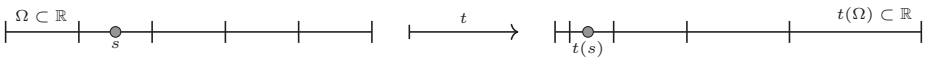


Figure V.3: The effect of reparameterizing the parameter space for a curve, where $t(s) = s^2,\ s \in [0, 1]$

the simplex surface, where parametric mappings are always bijective.) Bijective barycentric mapping between arbitrary polygons for any type of GBC can only be guaranteed if the polygons are convex and simple.

V.6.4.1 Mean value coordinates

Throughout the construction, we utilize mean value coordinates (MVCs) [HF06]. They are directly evaluable, but the potentially negative weights and non-bijective mappings limit their applicability to non-convex domains.

To avoid possible artifacts resulting from the MVC, and since barycentric coordinates are not the main focal point for the current work, the construction is limited to convex parametric polygon domains for the blending spline patches, see Section V.10.1.

[Flo15] is a good starting resource for generalized barycentric coordinates. For an extended read on GBC-related bijective mappings, we refer to [Sch17].

V.6.4.2 Parametric polygon domain

The domain of a polygonal surface is represented as a best-projection of the surface patch boundary into a planar polygon [SV18]. The best projection can be determined in various ways depending on the surface patch in question. Parametric polygon domains must be generated for the following components in a blending surface

- local geometry patches (see Section V.8.1),
- local sub-geometry patches (Section V.8.2), and
- blending spline patches (Section V.8.5).

V.6.4.3 Side-based parameterization

We adopt the parameterization of the GB patch to utilize techniques for tensor-product surfaces on polygon-based surfaces. The side based parameterization for a polygon $\Omega \subset \mathbb{R}^2$ is defined in [SV18; VSK17] as (s_i, h_i), where

$$s_i(v) = \frac{\lambda_i}{\lambda_{i-1} + \lambda i},$$
$$h_i(v) = 1 - \lambda_{i-1} - \lambda i,$$

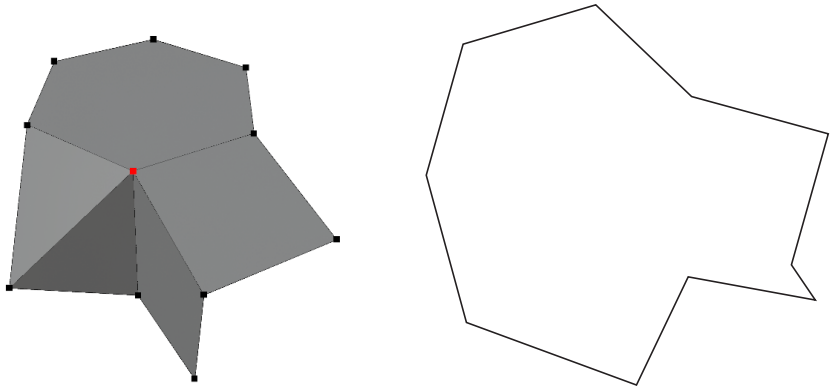


Figure V.4: 2D projection of a polygon surface boundary vertices. Left: a polygon's local control mesh. Right: the resulting planar polygon projection.

and $\lambda_i = \lambda_i(v)$ are generalized barycentric coordinates over the polygon Ω with sides $i = 1 \dots n$. The parameters s_i and h_i are known as the side- and distance parameters, respectively. The side-parameter s_i varies along side i, between 0 (on side $i-1$) and 1 (on side $i+1$). The distance-parameter h_i vanishes on side i, increases linearly along the adjacent sides $i-1$ and $i+1$ and evaluates to 1 on the "distant" sides. Mapping from a point, $v \in \Omega$, to side-based parameters (s_i, h_i) via generalized barycentric coordinates is illustrated in Figure V.5.

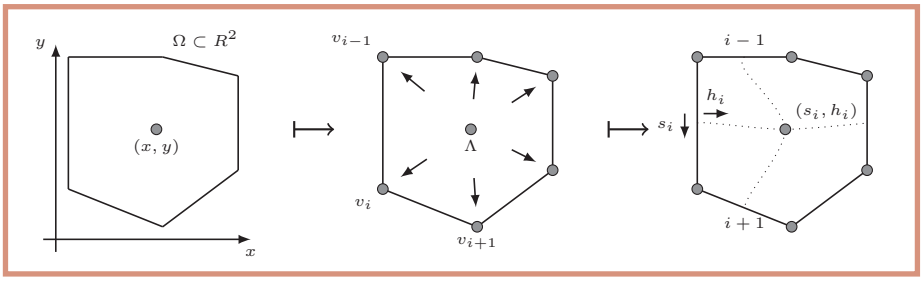


Figure V.5: Steps related to generation of sidewise parameters from a point in a planar polygon domain. Left-to-right: Parametric domain ↦ GBC ↦ sidewise coordinates.

V.6.4.4 Parametric polygon domain patchworks

As stated above, the GBCs lead to non-continuous constant parameter lines between adjacent blending spline patches. We propose to solve this problem with reparameterization by mapping the parameter space of the local geometry patch

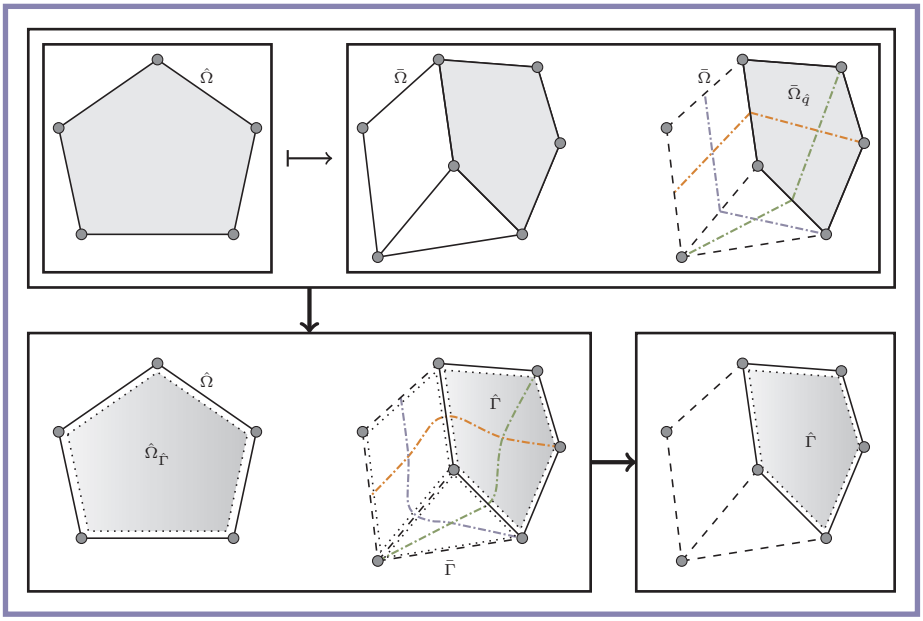


Figure V.6: Reparameterization through surface patchwork. Upper left: the best-projection parametric polygon for a blending spline patch boundary, where the vertices are the interpolation points of the respective local surfaces. Upper right: the best-projection local surface parametric polygon. The right-most polygon indicates the local sub-surface domain, which logically overlaps the parametric polygon of a blending spline patch. Lower left: reparameterization of the parametric polygon of the blending spline patch (left) to the parametric polygon of the overlapping local sub-patch (right) through a mapping, $\mathbb{R}^2 \mapsto \mathbb{R}^2$, which is the chosen patch construction, e.g., transfinite GB patch. The inset polygons represent the reparameterization mapping. The colored lines represent the resulting inner cross-edge constant parameter lines. Lower right: the reparameterized local sub-polygon parameter space polygon.

through an embedded patchwork, of which each patch is a sub-patch controlling the inner boundaries of the local patch illustrated in Figure V.6. This mapping is denoted $\hat{\Gamma}$ and is used in the definition of the local sub-patch in Section V.8.2. Parametric polygon domain restrictions are addressed in Section V.10.1, and an example of how to construct such patchworks is provided in Section V.8.3.

V.6.4.5 Spline knot graph

In contrast to tensor-products, where global tensor-lines (or parameter axis) can be defined, the proposed surface is represented on a polygon control mesh, where extended spline control is needed. Spline control is discussed

throughout the local surface and blending spline patch sections, V.8.1 and V.8, respectively. Instantiation of traditional spline knot vectors is explicitly discussed in Section V.7.6 and are used to scale the blending basis in Section V.8.4.2.

V.7 Topology, partitioning and spline knots

With tensor product blending spline surfaces, every single knot is associated with a geometry coefficient (which is a local surface). The topological layout is trivial since the order of the knots and knot multiplicities can be represented by a simple knot vector for each of the two parametric directions. This holds for both the coefficient space and the parametric space.

The polygonal surface introduces non-trivial relations for coinciding parametric spaces as well as the continuity between blending spline patches. We utilize well-defined circuits on top of a well-defined half-edge data structure to ensure consistent indexing of the knots. This is well known from graph theory; however, the concepts are described briefly to point out the methodology. First, we define four categories of domain topology (global, patch cover, local cover, and local sub-cover), then we present consistent circuit indexing before we finally describe how localized knot vectors can be deduced.

V.7.1 Global surface construction topology

Consider the blending spline surface as a composite patchwork over its adjacent knot intervals as outlined in Section 2.6.1. The topology of this composite patchwork can be interpreted through a half-edge structure, Section V.6.2.1. Q and $q_i \in Q$ denotes the collection of all knots (logical vertices) and individual knots, respectively, as visualized in Figure V.7.

V.7.2 Blending spline patch topology

The logical domain of each patch in the blending spline surface patchwork is equivalent to a face, f, in the half-edge structure. In this paper, we define a patch cover (denoted $\hat{Q}$ and illustrated in Figure V.7) as follows

$$\hat{Q} = \{\hat{q}_0, \ldots, \hat{q}_{m-1}\} \subseteq Q,$$

where m corresponds to the face valence (number of sides of the patch), e.g., for a quadrangular patch cover, $m = 4$, and $\hat{q}$ denotes its adjacent vertices.

V.7.3 Local surface topology

With each knot q, we associate a local surface whose logical domain is denoted $\bar{Q}$. The local domain is referred to as the local surface cover and is defined by the one-polygon neighborhood of q as follows

$$\bar{Q} = \{\bar{q}_0, \ldots, \bar{q}_{n-1}\} \subseteq Q, \tag{V.1}$$

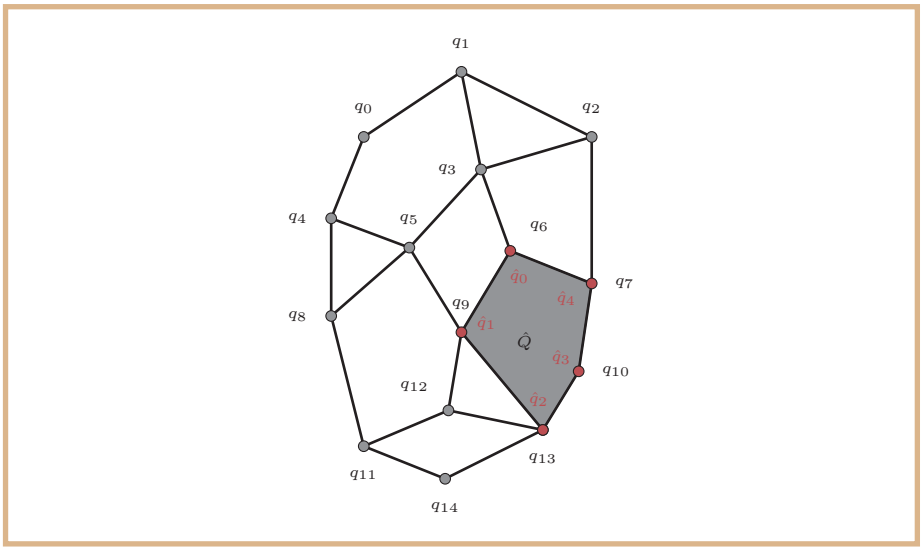


Figure V.7: The blending spline patch cover $\hat{Q} = \{q_6, q_9, q_{13}, q_{10}, q_7\}$.

where n corresponds to the valence of q, and $\bar{q}$ denotes the ordered set of adjacent vertices of the adjacent faces of q, hence the one-polygon neighborhood. The local surface coves can be of three different configurations: internal, boundary, or corner. Illustrated in Figure V.8. The internal type does not include the associated vertex as part of the local surface cover (boundary). Both the boundary and corner types, on the other hand, do include the associated vertex as part of their respective local surface covers. However, the corner type does not span any internal edges. If q is not a part of the local surface cover $\bar{Q}$, i.e., if q is in the interior of $\bar{Q}$, then the center vertex $\bar{q}_c = q$ is associated with $\bar{Q}$.

V.7.4 Local sub-surface topology

To be able to express the relationship between local surfaces and the blending spline patches, we provide notation for and definition of local sub-surface topology. Given a local surface cover, $\bar{Q}$, and an overlapping patch cover, $\hat{Q}$, associated with a vertex, $\hat{q}$, we refer to the local sub-surface cover as follows (illustrated in Figure V.9)

$$\bar{Q}_{\hat{q}} = \{\bar{q}_{\hat{q}_0}, \ldots, \bar{q}_{\hat{q}_{m-1}}\} \subseteq \{\hat{Q} \cap \{\bar{Q} \cup \bar{q}_c\}\},$$

where $\bar{q}_{\hat{q}} \in \hat{Q} \cap \{\bar{Q} \cup \bar{q}_c\}$.

V.7.5 Consistent indexing

The indexing rules need to be consistent when defining coordinate systems (parameter-space coordinate axis) which are common to the blending spline

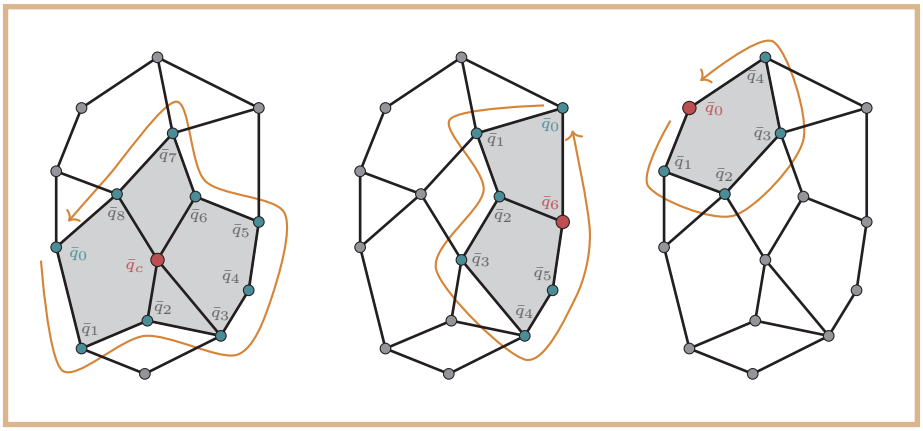


Figure V.8: Left-to-right: the three different configurations of local surface covers; internal, boundary, and corner. The internal type does not include the associated vertex as part of its local surface cover (boundary). The boundary and corner types, on the other hand, do include the associated vertex as part of their respective local surface covers. However, the corner type does not span any internal edges. The arrow paths denote local surface patch cycles.

patch and the blended local sub-surfaces. The graph circuits shared by the half-edge structure and the topological sub-cover definitions of $\hat{Q}$, $\bar{Q}$ and $\bar{Q}_{\hat{q}}$, provide consistent indexing.

V.7.5.1 Circuits

By utilising the directed half-edges, H, we can define counterclockwise circuits on the individual topological sub-covers, $\hat{Q}$, $\bar{Q}$, and $\bar{Q}_{\hat{q}}$, where $\hat{q}_0$, $\bar{q}_0$, and $\bar{Q}_{\hat{q}_0}$ represent the start of the individual circuits.

V.7.5.2 Blending spline centric circuits

Consider a blending patch cover and a set of overlapping local sub-surface covers. Its local coordinate systems are synchronized by rotation with respect to a chosen vertex in the blending spline cover, $\hat{q}_i$, and the respective vertex in the overlapping local sub-surface covers, $\{\bar{Q}_{\hat{q}_0} \dots \bar{Q}_{\hat{q}_{n-1}}\}$.

The overlapping local sub-surface cover circuits are co-aligned as follows. Firstly we choose one patch cover and select its circuit start vertex. Then, for each of the remaining local sub-surface cover circuits; we cycle through the vertices (candidates for start vertex) until the patch cover matches the chosen patch cover start vertex.

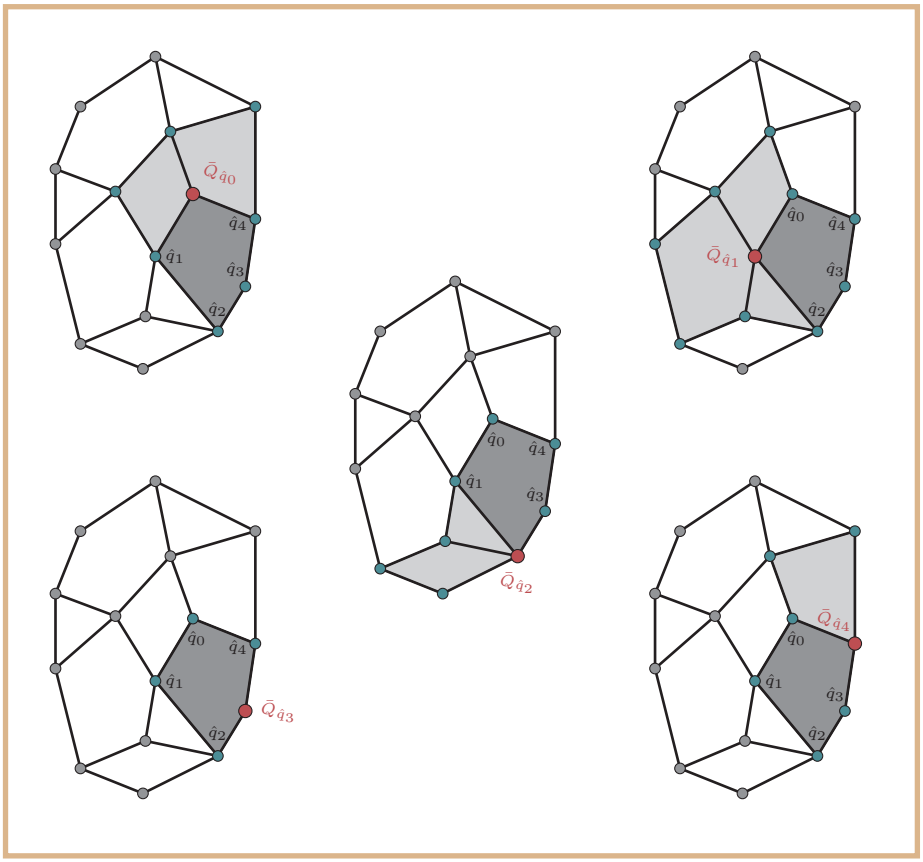


Figure V.9: The five local surface covers, $\bar{Q}_{\hat{q}_0}, \ldots, \bar{Q}_{\hat{q}_4}$, associated with the vertices of the patch cover $\hat{Q} = \{q_6, q_9, q_{13}, q_{10}, q_7\}$.

Example V.7.1 (Consistent local sub-surface covers)**.** Consider the topology in Figure V.7. Given a patch cover

$$\hat{Q} = \{\hat{q}_0, \ldots, \hat{q}_4\} = \{q_6, q_9, q_{13}, q_{10}, q_7\} \subset Q,$$

where q_6 is chosen as the common knot of the patch cover circuit, $\hat{q}_0 = q_6$, and the adjacent overlapping local surface covers $\bar{Q}$:

$$\begin{aligned}
\bar{Q}_{q_6} &= \{q_5, q_9, q_{13}, q_{10}, q_7, q_2, q_3, \bar{q}_c = q_6\},\\
\bar{Q}_{q_9} &= \{q_8, q_{11}, q_{12}, q_{13}, q_{10}, q_7, q_6, q_3, q_5, \bar{q}_c = q_9\},\\
\bar{Q}_{q_{13}} &= \{q_{13}, q_{10}, q_7, q_6, q_9, q_{12}, q_{11}, q_{14}\},\\
\bar{Q}_{q_{10}} &= \{q_{10}, q_7, q_6, q_9, q_{13}\},\\
\bar{Q}_{q_7} &= \{q_7, q_2, q_3, q_6, q_9, q_{13}, q_{10}\}.
\end{aligned}$$

Through cyclic indexing of the local surface cover circuits, aligning up at $\hat{q}_0$, we express the local sub-surface covers as follows

$$\begin{aligned}
\bar{Q}_{\hat{q}_0=q_6} &= \{\bar{q}_c, \bar{q}_1, \bar{q}_2, \bar{q}_3, \bar{q}_4\} = \{q_6, q_9, q_{13}, q_{10}, q_7\},\\
\bar{Q}_{\hat{q}_1=q_9} &= \{\bar{q}_6, \bar{q}_c, \bar{q}_3, \bar{q}_4, \bar{q}_5\} = \{q_6, q_9, q_{13}, q_{10}, q_7\},\\
\bar{Q}_{\hat{q}_2=q_{13}} &= \{\bar{q}_3, \bar{q}_4, \bar{q}_0, \bar{q}_1, \bar{q}_2\} = \{q_6, q_9, q_{13}, q_{10}, q_7\},\\
\bar{Q}_{\hat{q}_3=q_{10}} &= \{\bar{q}_2, \bar{q}_3, \bar{q}_4, \bar{q}_0, \bar{q}_1\} = \{q_6, q_9, q_{13}, q_{10}, q_7\},\\
\bar{Q}_{\hat{q}_4=q_6} &= \{\bar{q}_3, \bar{q}_4, \bar{q}_5, \bar{q}_6, \bar{q}_0\} = \{q_6, q_9, q_{13}, q_{10}, q_7\}.
\end{aligned}$$

V.7.6 Implicit knot vectors

There exist no intrinsic global parametric axes that can deduce a set of simple knot vectors since constructions with arbitrary topology are irregular on a hypothetical graph-based topology. Our approach to determining knot vectors is by describing spline knots as meta-information associated with the graph structure's vertices and edges. Consequently, implicit knot paths can be deduced from the graph before the conceptual instantiation of the blending spline patchwork.

V.7.6.1 Knot graph

The knot-centric graph meta-information, $\Pi = (\Pi^\sigma, \Pi^\phi)$ (see Section V.6.2.1), which contains a set of knot multiplicities Π^σ and a set of parametric edge distances Π^ϕ, defines a knot graph. A simple example showing knot graph notation is illustrated in Figure V.10.

Every $\pi^\sigma \in \Pi^\sigma$ describes a positive integer multiplicity associated with one vertex $q \in Q$, hence $|\Pi^\sigma| = |Q|$. $\pi^\sigma \in \{1, 2\}$ as a consequence of property $\mathcal{G}1$, Section 2.4 (minimal support). Each $\pi^\phi \in \Pi^\phi$ is a real number that describes a distance factor for an edge $e \in E$, hence $|\Pi^\phi| = |E|$. $\pi^\phi > 0$ due to property $\mathcal{G}1$, Section 2.4 (positivity).

V.7.6.2 Implicit parameter lines

An implicit parameter line is defined as the constant parametric line across the common edge of two blending spline patches, as illustrated on the right of Figure V.10.

V.7.6.3 An implicit knot path

Consider a path on the knot graph

$$Q^T = \{q_0, q_1, \dots\}.$$

The path is called a knot path if it follows a single implicit parameter line [LB15; Sed+03] (an implicit knot vector) on the topology. Multiplicity- and distance-factors of the k-th vertex along the knot path are expressed as

$$\Pi^\sigma_{q_k} = \Pi^\sigma\left(q_k\right),$$

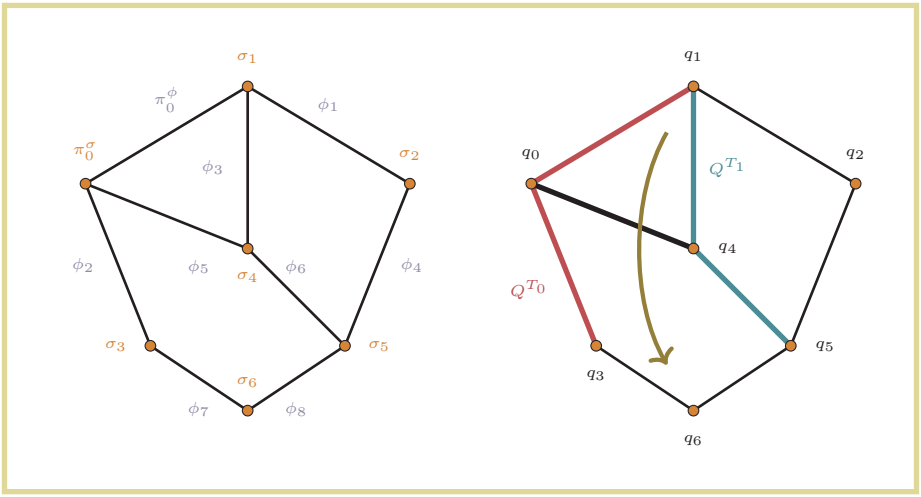


Figure V.10: Knot graph notation. Left: vertex multiplicity factors are enumerated with the notation π_i^σ and parametric edge distances are enumerated with notation π_j^ϕ (σ_i and ϕ_j used for simplicity). Right: shows two parallel paths, T_0 and T_1, on the graph.

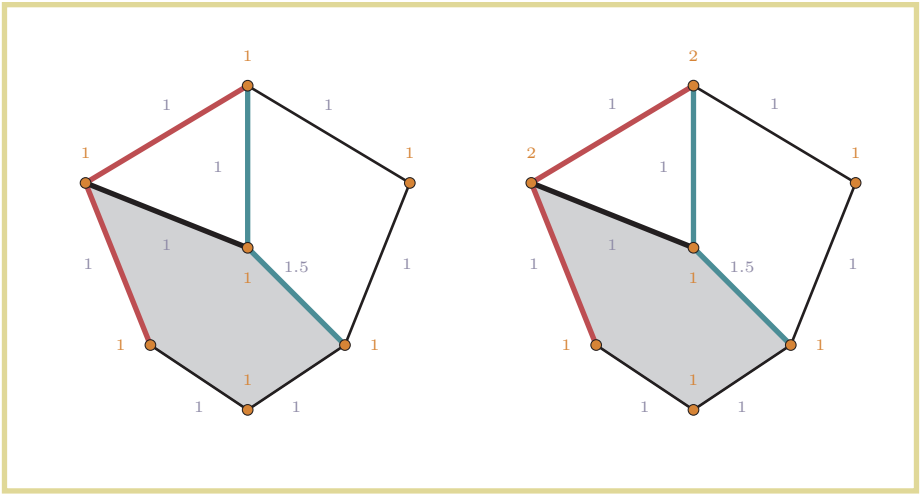


Figure V.11: Shows two knot graph configurations (both over the knot graph in Figure V.10) giving the multiplicity factors and edgewise parametric distances. Left: a configuration with single multiplicity knots. Right: a configuration with double multiplicity knots in q_0 and q_1.

and

$$\Pi^{\phi}_{q_k} = \Pi^{\phi}\left(e\left(q_{k-1}, q_k\right)\right),$$

respectively, and exploit the knot vector

$$T = \{t_j\}_{j=0}^{|T|-1},$$

where $|T| = \sum_{i=1}^{|Q^T|-2} \Pi^{\sigma}_{q_i} + 2$ and t_j is a one- or two-tuple of knot values. Furthermore, generation rules for a (strictly) increasing knot vector are defined as follows

- the initial double-knot takes the value zero,
- the values for the subsequent knots are obtained by accumulating the distance factors Π^{ϕ} and utilizing the knot multiplicities Π^{ϕ}, which are both found via collecting the tuples Π along the knot path Q^T, and
- the values for the double-knot at the end are found analogously, by adding the last distance factor to the second to last knot value.

Likewise,

$$\begin{aligned} t_0 &= \{0, 0\}, \\ t_k &= \begin{cases} \{\cdot\}, & \text{if } \Pi^{\sigma} = 1, \\ \{\cdot, \cdot\}, & \text{if } \Pi^{\sigma} = 2, \end{cases} \text{ with } \cdot = t_{k-1}[-1] + \Pi^{\phi}_{q_k}, \\ t_{-1} &= \{\cdot, \cdot\} \text{ with } \cdot = t_{-2}[-1] + \Pi^{\phi}_{q_{-1}}, \end{aligned}$$

where k indexes the inserted knot tuple in T (indexed over Q^T), and the notation $[-x]$ and t_{-x} denote reverse indexing (from the rear) of a tuple or a set, respectively. The final spline knot vector is obtained by collecting the numbers from tuples $t_j \in T$ in order.

V.7.6.4 Explicit knot vectors

The transition from implicit knot paths to explicit knot vectors can be delineated to a set of implicit knot vectors along the outer edges of single patch covers, thanks to property $\mathcal{P}1$, Section 2.3.2 (minimal support). The inferred knot definition can then be expressed as a limited lookup function.

Given the implicit global knot path Q^T, for one knot interval $[q_k, q_{k+1}] = [\hat{q}_{i-1}, \hat{q}_i]$, with the relative knot-centric indices $[0, 1]$. A strictly increasing knot vector T along one specific local knot interval is defined as follows

$$t_0 = \begin{cases} \{0, 0\}, & \text{if } \Pi^{\sigma}_{q_k} = 2, \text{ or } q_k \text{ is an edge}, \\ \left\{0, \Pi^{\phi}_{q_k}\right\}, & \text{otherwise}, \end{cases}$$

and

$$t_1 = \begin{cases} \{\cdot, \cdot\}, & \text{if } \Pi^{\sigma}_{q_{k+1}} = 2, \\ & \text{or } q_{k+1} \text{ is an edge}, \\ \left\{\cdot, \cdot + \Pi^{\phi}_{q_{k+2}}\right\}, & \text{otherwise}, \end{cases}$$

with $\cdot = t_0[-1] + \Pi^{\phi}_{q_{k+1}}$.

V.7.6.5 Patch-cover-wise knot vector notation

Knot vector values are assumed to be accessed by cycling over the polygon sides. A single interval knot vector with respect to side i (counterclockwise) on a patch cover, $\hat{Q}$, yields

$$\hat{T}_{e_i} = \{\hat{T}_{i-1}, \hat{T}_i\} = \{\{\hat{t}_{i-1,0}, \hat{t}_{i-1,1}\}, \{\hat{t}_{i,0}, \hat{t}_{i,1}\}\}. \tag{V.2}$$

The following example shows the expansion of explicit knot vectors from implicit knot paths.

Example V.7.2. Consider the knot graph configuration illustrated on the left of Figure V.11. It follows that

$$\begin{aligned}\Pi &= \left(\Pi^{\sigma}, \Pi^{\phi}\right) \\ &= \left(\begin{array}{l}\left\{\pi_0^{\sigma}, \pi_1^{\sigma}, \pi_2^{\sigma}, \pi_3^{\sigma}, \pi_4^{\sigma}, \pi_5^{\sigma}, \pi_6^{\sigma}\right\}, \\ \left\{\pi_0^{\phi}, \pi_1^{\phi}, \pi_2^{\phi}, \pi_3^{\phi}, \pi_4^{\phi}, \pi_5^{\phi}, \pi_6^{\phi}, \pi_7^{\phi}, \pi_8^{\phi}\right\}\end{array}\right) \\ &= \left(\begin{array}{l}\{1,1,1,1,1,1,1\}, \\ \{1,1,1,1,1,1,1.5,1,1\}\end{array}\right).\end{aligned}$$

Expanding a knot vector over the entire path Q^{T_0} on Figure V.11 yields $T = \{\{0,0\}, 1, \{2,2\}\}$, while expanding the same path over one single interval $e = [q_0, q_3]$ yields $T_e = \{T_{q_0}, T_{q_3}\} = \{\{0,1\}, \{2,2\}\} = \{0,1,2,2\}$.

Example V.7.3. Considering the same knot graph configuration as in Example V.7.2, the knot expansion with respect to side $e_i = [q_4, q_0]$ on patch cover $\hat{Q} = \{q4, q0, q3, q6, q5\}$, which would expand along both paths Q^{T_0} and Q^{T_1}, would yield

$$\begin{aligned}\hat{T}_{e_i} &= \left\{\hat{T}_{i-1}, \hat{T}_i\right\} = \left\{T_{e=[q0,q3]}, T_{e=[q4,q5]}\right\} \\ &= \left\{\begin{array}{l}\{0,1,2,2\}, \\ \{0,1,2.5,2.5\}\end{array}\right\}.\end{aligned}$$

V.7.6.6 Multiple (inner) knots

Again, due to the minimal support property ($\mathcal{P}1$, Section 2.3.2) and the restriction to a single knot interval, we consider either single or double knots. Any knot multiplicity higher than two would result in disjoint knots (associated with disjoint local geometry patches), which would have no impact when seen from the perspective of a blending spline patch. Since the derivatives of the B-function vanishes at the knots, the spline surface inherits the continuity properties from the associated local surface(s). As a result, single knots allow us to control both the G and the potential C continuities, whereas double knots allow us to control the potential C continuity while possessing G^0 continuity.

Example V.7.4. Consider the knot graph configuration with multiple knots as illustrated on the right of Figure V.11

$$\begin{aligned}\Pi &= \left(\Pi^{\sigma}, \Pi^{\phi}\right) \\ &= \left(\begin{array}{l} \left\{\pi_0^{\sigma}, \pi_1^{\sigma}, \pi_2^{\sigma}, \pi_3^{\sigma}, \pi_4^{\sigma}, \pi_5^{\sigma}, \pi_6^{\sigma}\right\}, \\ \left\{\pi_0^{\phi}, \pi_1^{\phi}, \pi_2^{\phi}, \pi_3^{\phi}, \pi_4^{\phi}, \pi_5^{\phi}, \pi_6^{\phi}, \pi_7^{\phi}, \pi_8^{\phi}\right\} \end{array} \right) \\ &= \left(\begin{array}{l} \{2,2,1,1,1,1,1\}, \\ \{1,1,1,1,1,1,1.5,1,1\} \end{array} \right).\end{aligned}$$

Expanding a knot vector over the entire path Q^{T_0} on Figure V.11 yields $T = \{\{0,0\},\{1,1\},\{2,2\}\}$, while expanding the same path over one single interval $e = [q_0, q_3]$ yields $T_e = \{T_{q_0}, T_{q_3}\} = \{\{0,0\},\{1,1\}\} = \{0,0,1,1\}$.

Hence, patch cover knot expansion equivalent to Example V.7.3 would yield

$$\begin{aligned}\hat{T}_{e_i} &= \left\{\hat{T}_{i-1}, \hat{T}_i\right\} = \left\{T_{e=[q0,q3]}, T_{e=[q4,q5]}\right\} \\ &= \left\{ \begin{array}{l} \{0,0,1,1\}, \\ \{0,1,2.5,2.5\} \end{array} \right\}.\end{aligned}$$

V.8 Blending spline surface

The hierarchical and topological relationships of the blending spline surface construction were defined in Section V.7. This section considers the realization of the surface construction.

As described in Section 2.6.1, a blending spline surface can be interpreted as a patchwork. In short: a blending spline surface, S, can be interpreted as a piecewise composition of adjacent blending spline patches, where a global blending spline patch, $\hat{S}$, is topologically defined over the blending spline patch cover, $\hat{Q}$. $\hat{S}$ is realized by blending together evaluations of local sub-surfaces, $\hat{L}$, defined over the local sub-surface covers, $\bar{Q}_{\hat{q}}$. $\hat{L}$ are sub-surfaces in their respective local surfaces, $\bar{L}$, defined over local surface covers, $\bar{Q}$, which is associated with respective vertices, $\hat{q}$, of the blending spline patch.

The blending spline patch formulation, which facilitates the extension from tensor-products to surfaces based on N-sided polygons, is derived and presented in the following.

V.8.1 Local surfaces

Blending splines [DBL09; DLB06] extend the notion of B-spline coefficients from spatial points to point- or vector evaluated functions. With surface constructions, the coefficients associated with the knots are local surfaces, $\bar{L}$, and associated spatial frames. In the case of a surface based on an arbitrary poly-mesh topology, this means that the local surface, associated with a vertex q, covers the local surface cover, $\bar{Q}_q$. It possesses a planar parametric polygon domain $\bar{\Omega}_q \subset \mathbb{R}^2$ with

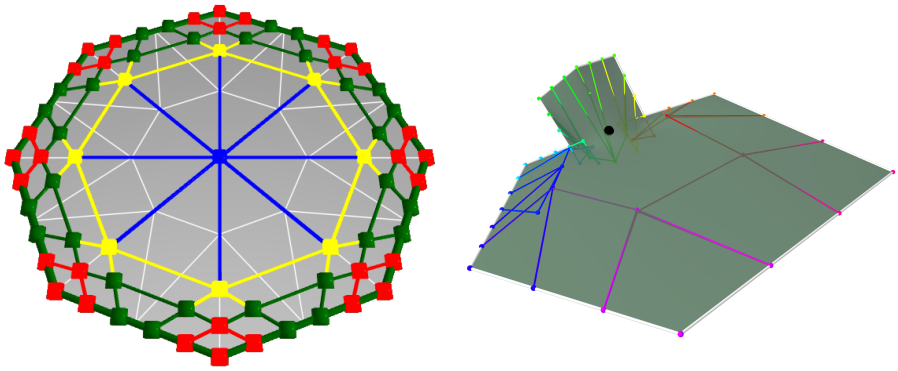


Figure V.12: Left: A GB-patch of valence 8 over a convex domain. Right: A GB-patch of valence 10 over a concave domain.

local surface cover associated barycentric coordinates $\bar{\Lambda}$, where $\bar{\lambda}_i$ is associated with $\bar{q}_i$.

The type of local surface geometry can be chosen with respect to the application at hand and can be any polygonal surface construction adhering to the criteria above. A ribbon-based variation of the generalized Bézier patch (GB-patch) presented in [SV18] was chosen in the present work, mainly for two reasons. Firstly, they support both convex and concave polygonal parametric domains. Secondly, they are direct extensions of tensor-product Bézier-type surfaces utilized in the original blending spline tensor-product surfaces. Their well-established properties with respect to utilization in surface modeling were the main criteria and as well as being the de facto industry standard.

Other alternatives, which we do not consider here, can be utilized; S-patches [LD89], or, in special cases, triangular or quadrilateral Bézier surface patches, sub-polygon surfaces on tensor-product patches, or cage-constructions, to name a few.

V.8.1.1 GB-patches as local geometry

The evolution of GB-patches is covered through a series of papers, including [SV18; VSK16; VSK17]. Two examples (convex and concave) are illustrated in Figure V.12. A ribbon-based GB-patch is defined over a simple polygonal domain, $\bar{\Omega} \subset \mathbb{R}^2$ with points on the domain $\bar{v} \in \bar{\Omega}$, as

$$\bar{L}(\bar{v}) = \frac{1}{b_{\text{sum}}(\bar{v})} \sum_{i=1}^{n} R_i(s_i, h_i), \tag{V.3}$$

where $R_i(s_i, h_i)$ are side ribbons, and

$$b_{\text{sum}}(\bar{v}) = \sum_{i=1}^{n} \sum_{j=0}^{d_i} \sum_{k=0}^{l_i-1} \mu_{j,k}^{i} b_{j,k}^{i}(s_i, h_i),$$

is a basis normalization function.

The i-th side ribbon is defined over a side-based parameterization as

$$R_i(s_i, h_i) = \sum_{j=0}^{d_i} \sum_{k=0}^{l_i-1} C_{j,k}^{i} \cdot \mu_{j,k}^{i} \; b_{j,k}^{i}(s_i, h_i),$$

where

$$b_{j,k}^{i}(s_i, h_i) = b_j^{d_i}(s_i) b_k^{2l_i-1}(h_i),$$

is the bi-variate Bernstein basis associated with the control point $C_{j,k}^{i}$, and

$$\mu_{j,k}^{i} = \begin{cases} \alpha_i = h_{i-1}^2/(h_{i-1}^2 + h_i^2), & \text{when } 2j < d, \\ 1, & \text{when } 2j = d, \\ \beta_i = h_{i+1}^2/(h_{i+1}^2 + h_i^2), & \text{when } 2j > d, \end{cases}$$

is a rational side-distance scalar weight.

V.8.2 Local sub-surfaces

The local sub-surface, $\hat{L}$, is defined as a sub-surface construction $\bar{L}(\hat{v}) \subset \mathbb{R}^3$, over the topological domain $\bar{Q}_{\hat{q}} = \hat{Q}$ of the local surface construction associated with $\hat{q}$.

The parametric coordinates $\hat{v} \in \hat{\Omega}$ are the planar coordinates of the blending spline patch parametric polygon, mapped into the parametric subdomain of the local surface. The continuity of the global blending construction is controlled by the distribution of this parameterization across the internal edge of the parametric polygon domain $\bar{\Omega}$ of the local surface. This sub-polygon mapping is required to be a diffeomorphism in order to preserve continuity properties across the internal local surface edges associated with the blending spline patch boundaries.

To construct $\hat{L}(\hat{v})$ under this requirement, we propose to generate the parametric polygon mapping, $\hat{v} \in \hat{\Omega} \mapsto \hat{v} \in \bar{\Omega}_{\hat{q}}$, utilizing a custom patchwork embedded in the parametric polygon $\bar{\Omega}$ of the local surface construction $\bar{L}$. The patchwork is denoted $\bar{\Gamma} \hookrightarrow \bar{\Omega}$ while one patch in the patchwork, over the respective patch cover domain, is denoted $\hat{\Gamma}(\hat{v}) \hookrightarrow \bar{\Omega}_{\hat{q}}$. Then, by utilizing the mapping

$$f : \hat{\Omega} \mapsto \hat{\Gamma}, \tag{V.4}$$

and letting

$$\hat{v}_{\hat{\gamma}} = \hat{\Gamma}(\hat{v}),$$

where both $\hat{v}_{\hat{\gamma}}$ and $\hat{v}$ are points in the $\mathbb{R}^2$ domain bounded by $\hat{\Omega}$, we obtain the local sub-polygon surface

$$\hat{L}(\hat{v}) = \bar{L}(\hat{v}_{\hat{\gamma}}). \tag{V.5}$$

The mapping (V.4) is implicitly applied in Section V.8.4.3 and exemplified in the following.

V.8.3 Construction of the $\bar{\Gamma}$-patchwork

A patchwork in this context can be like any other patchwork. However, we are mapping $\mathbb{R}^2 \mapsto \mathbb{R}^2$, so the construction could potentially be simplified. An example of a suitable patch is a domain-scaled version of the ribbon-based generalization of the Coons surface [VRS11], which yields

$$\hat{\Gamma}(\hat{v}) = \sum_{j=1}^{n} \delta_j^{h_j} \; \hat{R}_j(s_j, h_j) \; \mu_j(h_1, \ldots, h_n),$$

where $\hat{R}_j$ is constructed to preserve $\bar{Q}$'s internal cross-edge continuity, and

$$\mu_j(h_1, \ldots, h_n) = \frac{D_j^n}{\sum_{k=1}^{n} D_k^n},$$

is a variation of the special side-blend basis [VRS11] with

$$D_{j1,j2,\ldots,jn}^n = \Pi_{j \neq j1,j2,\ldots,jn}^n \; \mathfrak{B}(h_j),$$

where $\mathfrak{B}(h)$ is a monotonous C^k-smooth B-function.

V.8.3.1 $\bar{\Gamma}$-patchwork ribbon construction

$\bar{\Gamma}$-surface patchworks are created posterior to the local geometry construction $\hat{L}$. We utilize ribbon-based patches, such as [SVR14]. Our ribbons are constructed by applying fairing rules to the $\bar{\Omega}$-domain [Kob00]. In the fairing-process, we use constant normalized ribbon tangents, which utilize normal-cages constructed in $\mathbb{R}^3$ by fairing counterclockwise along the internal boundary edges of the $\bar{\Omega} \subset \mathbb{R}^2$ domain, and uniform knot vectors. This leads to a $\hat{\Omega} \mapsto \hat{\Gamma}$ mapping that provides constant parameter lines over the internal edges of the sub-polygon blending constructions. A schematic depiction of the ribbons in the polygon domain $\bar{\Omega}$ of $\bar{L}$ can be seen in Figure V.13.

V.8.4 Sidewise blending

Tensor product surfaces blend along a pair of global parametric axes. The poly-mesh topology construction augments this behavior through a ribbon-based approach derived from transfinite surface interpolation [VRS11]. This translates into constructing sidewise ribbons and then blending these ribbons using a special sidewise poly B-function.

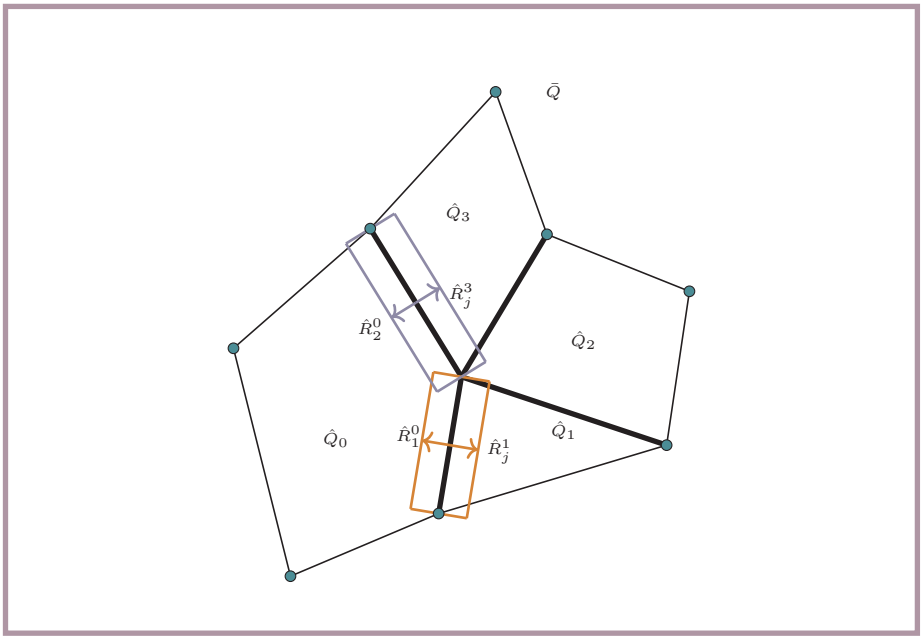


Figure V.13: $\bar{\Gamma}$ patchwork ribbons for the inner edges of $\bar{Q}$ between $\left\{\hat{Q}_0, \hat{Q}_3\right\}$ and $\left\{\hat{Q}_0, \hat{Q}_1\right\}$.

We start by defining the special sidewise poly B-function. Then we define domain scaling factors before constructing sidewise ribbons with respect to the spline knot vectors, and finally defining the blending spline patch and the composite blending spline surface.

V.8.4.1 Sidewise poly-B-function

C^k-smooth B-functions associated with the sides of an n-sided blending spline patch can be expressed as

$$\hat{B}_i = \sum_{i=1}^{n} \mathfrak{B}(s_i)\ \mu_i,$$

where

$$\mu_i = \frac{\widetilde{B}_i}{\sum_{j=1}^{n} \widetilde{B}_j},$$

are sidewise weight deficiency functions over the distance parameter h_i, with

$$\widetilde{B}_k = \prod_{i=1,\ i\neq k}^{n} \mathfrak{B}(h_i),$$

and $\mathfrak{B}$ is a monotonous C^k-smooth B-function. Note that

$$\sum_{i=1}^{n} \hat{B}_i = 1,$$

so the B-function blending basis possesses a partition of unity property within the parametric domain of the patch.

V.8.4.2 Sidewise scaling function

From classic blending spline theory, we obtain the global-to-local mapping functions and scaling factors ω [Lak07, theorem 2.4] and δ [Lak07, definition 2.7], respectively. In the present application, the global-to-local mapping functions are rendered superfluous, due to the local nature of barycentric coordinates, the minimal support of the local surface covers, and the sub-surface approach (Section V.8.2). As a result, the scaling function becomes a constant $\delta = \frac{1}{t_{k+1}-t_{k-1}}$.

By adopting the domain scaling-factors and using the patchwise knot vector notation defined in (V.2), sidewise scaling functions δ^h_{i-1} and δ^h_i for the knot vector $\hat{T}_{e_i}$ of side i can be defined

$$\delta^h_{i-1} = \frac{1}{\hat{t}_{i-1,k+1} - \hat{t}_{i-1,k-1}}, \qquad \text{where } k = \begin{cases} 1, & \text{if } 0 \le s_i < 1, \\ 2, & \text{otherwise.} \end{cases}$$
$$\delta^h_i = \frac{1}{\hat{t}_{i,k+1} - \hat{t}_{i,k-1}},$$

V.8.4.3 Side-based sub-polygon blending

An expression for blending two local sub-surfaces along one side parameter for a blending spline patch is established in the following. Given a local sub-surface, $\hat{L}_i$, associated with the i^{th} vertex, together with explicit sidewise scaling factors associated with the i^{th} side, and the respective sub-surface parametric polygon domain $\hat{v}_{\bar{\Omega},i}$, the blending along the i^{th} side parameter s_i for a blending spline patch can be expressed as follows

$$\begin{aligned} \hat{L}^s_i = (1 - \mathfrak{B}(s_i))\ \hat{L}_{i-1}(\hat{v}_{\bar{\Omega},i-1}, \delta^h_{i-1}) + \\ \mathfrak{B}(s_i)\ \hat{L}_i(\hat{v}_{\bar{\Omega},i}, \delta^h_i). \end{aligned} \tag{V.6}$$

V.8.5 The blending spline patch and composite surface

Finally, the blending spline patch can be expressed as

$$\hat{S}(\hat{v}) = \sum_{i=1}^{n} \hat{L}^s_i\ \mu_i. \tag{V.7}$$

The polygon spline surface is obtained by collecting all the blending spline patches $\hat{S}$ over $\hat{Q}$, defined on the poly-mesh topology G as

$$S(v) = \{\hat{S}_0(\hat{v}_0), \ldots, \hat{S}_{k-1}(\hat{v}_{k-1})\},$$

where k is the number of blending spline patches associated with Q.

V.9 Results

The novelty of the new construction is the global parameterization and how the continuity and smoothness of this are controlled. The requirements governing the need for this type of control is the desired utilization of a given set of local surface types, i.e., Bézier-type surfaces. These are simple, flexible and well-known (de facto industry standard). This again is governed by the potential freedom and control an artist has while utilizing them for free-form modeling or shape control through parameter line control.

To demonstrate the features and components of the construction, we utilize a small data set

$$\begin{aligned} Q &= \{q_0, \ldots, q_8\} \\ &= \{(2, 1.75, 5), (4, 2, 5), (0, 1.75, 4), \\ &\quad (2, 0, 3), (5, 0.5, 3), (0, 2, 2), \\ &\quad (5, 0.5, 2), (2, 1, 1), (3, 0.5, 1)\} \\ F &= \{\{q_0, q_3, q_2\}, \{q_0, q_1, q_4, q_3\} \\ &\quad \{q_4, q_6, q_8, q_7, q_3\}, \{q_2, q_3, q_7, q_5\}\}. \end{aligned}$$

Figure V.14 shows a shaded rendering together with the control poly-mesh used for approximation. The composite blending spline surface consists of four blending spline patches of valence three, four (x2), and five. They are blended from one central and eight boundary local surfaces with a valence ranging from two to four. All except one of the local surfaces possesses convex parametric polygon domains.

V.9.1 Local surface construction

Ribbon-based GB-patches [SV18] are used, with quadratic ribbons (equal-sided; $d = 3$, $l = 2$) , whose control points minimize a linear system of equations

$$\begin{aligned} \min &= \sum_i (\bar{L} \circ \bar{v} - \xi_i)^2, \\ \text{s.t. } &\bar{L} \circ \text{proj}_{\bar{\Omega}}\ p_c = \xi(p_c). \end{aligned}$$

The resulting local surface patches cover the data set, interpolate the associated vertex, and retain the features of the edges. A selection of local surfaces are illustrated in Figure V.15.

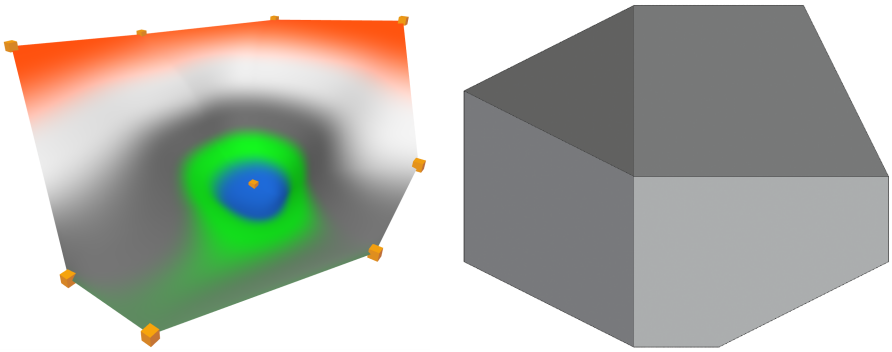


Figure V.14: A blending spline surface constructed over a small synthetic data set. The construction blends nine local polygon surfaces into four adjacent blending spline patches of three to five sides. Left: Rendered and shaded with a color gradient along the up-axis. The orange cubes represent (collapsed) local surfaces situated at their respective interpolation points. Right: Top-view rendering of the surface control poly-mesh.

V.9.1.1 Continuity

Figure V.16 shows a blending spline surface patchwork composition rendered with isophotes. Equally-scaled normals along the boundaries of each individual blending spline patch are highlighted. The isophote lines and the coinciding normals of adjacent patches along the surface's inner boundary demonstrate G^1-continuity between the spline patches.

Four zoomed top-down views of the surface with respect to the central vertex of the poly-mesh are shown in Figure V.17. The visualizations show constant parameter lines across the edges in the patchwork, shaded in the same color. They are generated from the sidewise parameter lines, $s(\hat{v})$, of the blending spline patches' respective adjacent sides. To provide a visual context, with respect to Figure V.16, the respective h-parameter lines are visualized in patchwise colors. The continuity of the underlying mapping, $\hat{\Omega} \mapsto \hat{\Gamma}$, is reflected in the continuity of the constant parameter lines, which is clearly at least G^1.

V.9.2 Arbitrary poly-mesh height map interpolation

Figure V.18 illustrates blending spline surfaces approximated from two different synthetic poly-mesh height maps. The vertices represent peaks, and the edges represent connecting ridge features. The resulting blending spline surface interpolates the vertices. The valence of the blending spline patches ranges from three to ten, and the valence and side count for the local surfaces range from two to five and three to twelve, respectively.

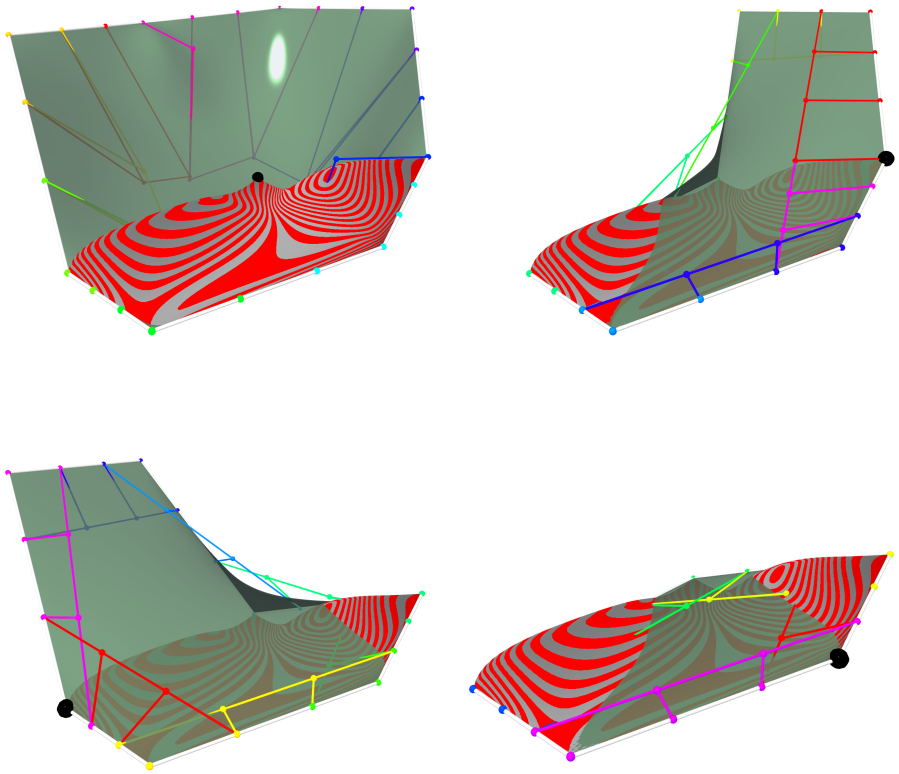


Figure V.15: Four of the five local surfaces contributing to the red 5-sided evaluation patch in Figure V.16. The local surfaces are transparently shaded in green together with the blending spline patch for reference. The fifth local surface (which is not shown) has the same shape as the one in the lower right, except its interpolation points are located one "knot" to the left (counterclockwise).

V.10 Concluding remarks

Blending spline constructions are generally C^k-smooth. The smoothness properties can be categorized in internal smoothness (which is bound by the basis function and the local surfaces) and smoothness at the knots. The construction possesses a Hermite interpolation property at the knots, which implies that the knot smoothness is inherited from the local geometry. In the case of tensor products, smoothness over the edges (between the knots) is inherited similarly since the tensor product parameter lines over the edges are constant and perpendicular to the edge.

However, a polygon-based blending spline surface is generally C^0 over the edges from the input mesh topology. In order to control and improve the inter-patch continuity, parameter lines that connect two adjacent patch covers across

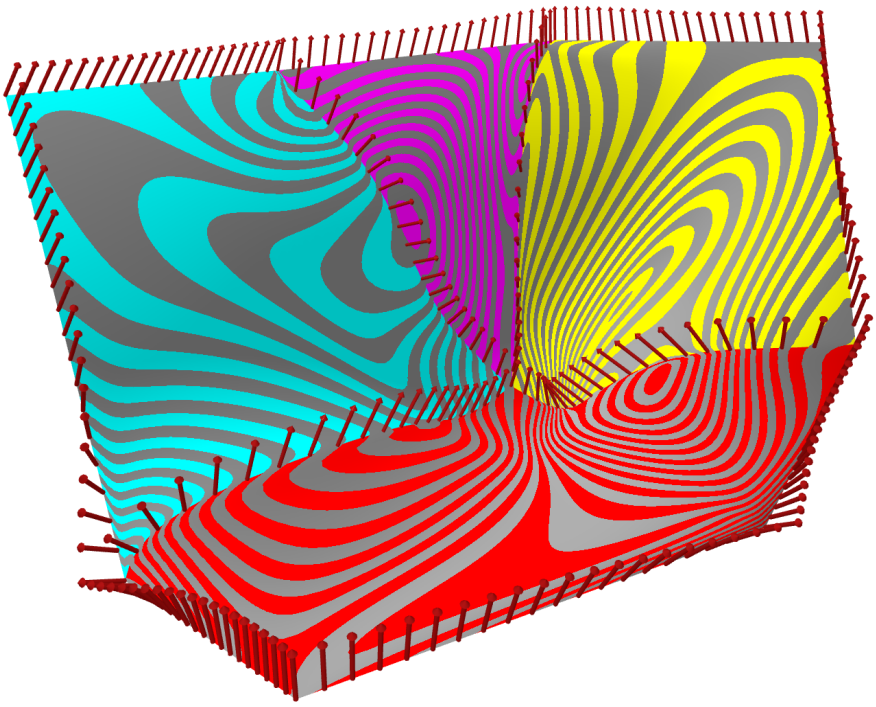


Figure V.16: Blending spline surface shaded with isophote lines. Isophote lines are colored with unique patchwise colors. Normals are visualized along each of the patch boundaries (normals overlap along the inner boundaries).

the common edge have to be continuous and smooth. Those lines can be required to be perpendicular to the edge at the boundary for consistency.

The first two criteria, internal smoothness and smoothness at the knots, are provided by the blending spline construction. Smoothness over the common edge is addressed in the following.

V.10.1 Domain restrictions

For a blending spline patch cover, $\hat{Q}$, there is an associated blending spline patch, $\hat{S}$, and a planar polygon, $\hat{\Omega}$. Furthermore, there are associated local surfaces, $\bar{L}_1, \ldots, \bar{L}_n$, each with its own associated planar polygon, $\bar{\Omega}_i$, $i = 1, \ldots, n$, respectively. Finally, there are local sub-surfaces, $\hat{L}_1, \ldots, \hat{L}_n$, with their respective planar sub-polygons $\hat{\Omega}_{\bar{\Omega}}$.

Between overlapping planar sub-polygons of blending spline patch domains, $\hat{\Omega}_{\bar{\Omega}} \mapsto \hat{\Omega}$, there is required to be a diffeomorphism in order to ensure desired smoothness between the patches. Two differentiable maps f and g, are defined via applying some restrictions as follows.

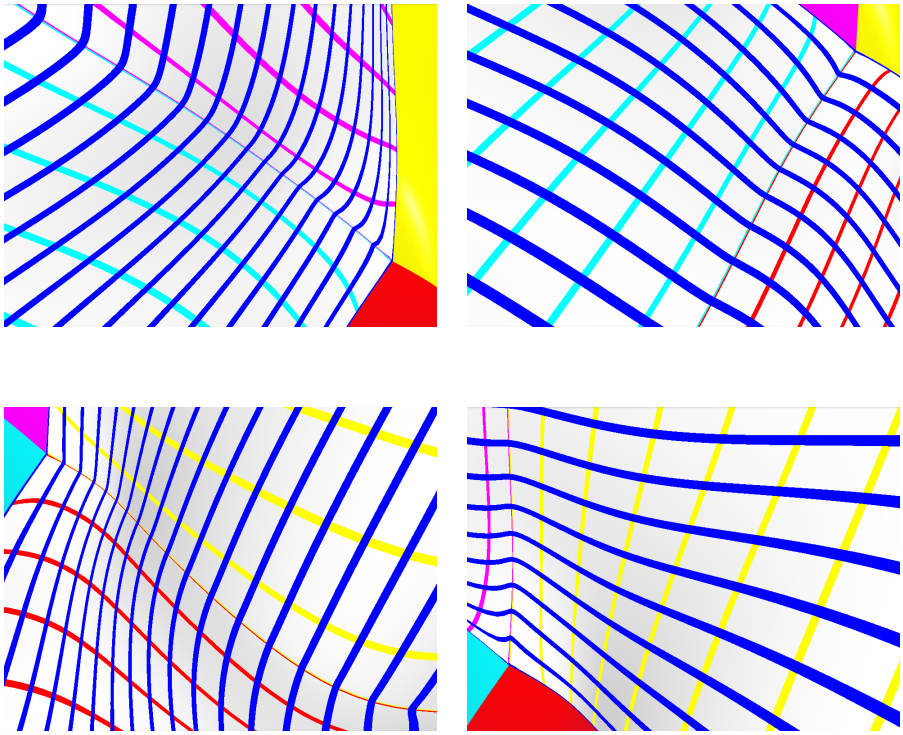


Figure V.17: A part of the surface in Figure V.16, visualized edgewise, using constant parameter lines for the adjacent s-parameter directions (Section V.6.4.3). The respective h-parameter lines along with the adjacent blending spline patches are rendered in the patches' colors.

1. Between the individual parametric polygons of the local sub-surfaces in a blending spline patch, we have that $\forall\{\hat{\Omega}_{\bar{\Omega},i}, \hat{\Omega}_{\bar{\Omega},j}\} \in \{\hat{\Omega}_{\bar{\Omega},1}, \ldots \hat{\Omega}_{\bar{\Omega},n}\}$, $\exists\, f : \hat{\Omega}_{\bar{\Omega},i} \mapsto \hat{\Omega}_{\bar{\Omega},j}$, such that $\hat{\Omega}_{\bar{\Omega},i} \cong \hat{\Omega}_{\bar{\Omega},j}$, where $i \neq j$.
2. Between the parametric polygons of a blending spline patch and an individual local sub-surface, we have that $\forall\{\hat{\Omega}_{\bar{\Omega},i}\} \in \{\hat{\Omega}_{\bar{\Omega},1}, \ldots \hat{\Omega}_{\bar{\Omega},n}\}$, $\exists\, g : \hat{\Omega} \mapsto \hat{\Omega}_{\bar{\Omega},i}$, such that $\hat{\Omega}_{\bar{\Omega},i} \cong \hat{\Omega}$.

Our solution is to achieve these domain boundary restrictions through the proposed patchwork solution. See illustration in Figure V.6 for visual aid.

V.10.2 Smoothness by a planar-polygon embedded patchwork

As described above, the internal cross-boundary smoothness of a blending spline patch, with respect to two adjacent sub-surfaces, can be addressed by assuming the presence of sufficiently restricted domains. Due to the minimal support

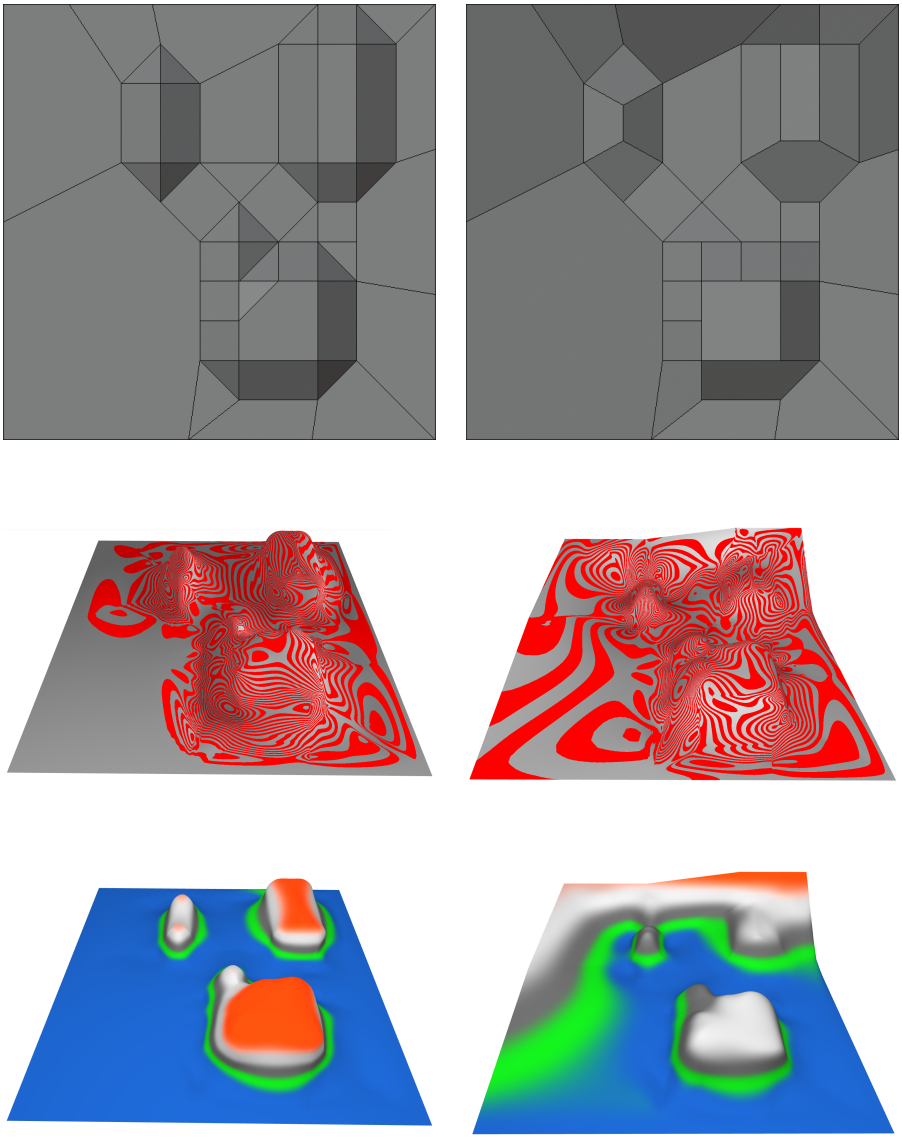


Figure V.18: Two different approximation examples. Top-to-bottom: poly-mesh height map, shaded rendering with isophote lines, shaded rendering using an up-axis color gradient.

property ($\mathcal{P}1$, Section 2.4), the edge-centric local surfaces $\bar{L}_i$ and $\bar{L}_j$ are the only non-zero local surface contributions over the edge $\hat{e} = [q_i, q_j]$. In the case of tensor-products, if the two associated local surfaces and the blending function are sufficiently smooth, the surface construction is smooth as well. This follows since the intrinsic parameter lines of a tensor-product surface are smooth across the edge.

However, in the irregular grid case (where we utilize generalized barycentric coordinates), the intrinsic cross-boundary parameter lines are not necessarily smooth (as pointed out in [Lak07]). A criterion for smoothness can be controlled by shaping of the intrinsic (edgewise) parameter lines for the associated local surfaces with respect to their local sub-surfaces. In this paper we have proposed a solution to embed a patchwork construction in the parametric polygon of the local surfaces. The parametric polygon of each of the local sub-surfaces is congruent with the parametric polygon of the blending spline patch, and the boundary coefficients of the individual Γ-patches are modeled from the shape and desired parametric directions of the parametric polygon domain of the local sub-surfaces. A realized construction where the resulting smoothness-contribution is controlled by applied fairing rules was discussed in Section V.8.3 Again, see the illustration in Figure V.6 for visual aid.

V.10.3 Implementation and processes

From an implementation point-of-view, the construction can be seen as being "implicit on construction" while being "explicit on evaluation". This refers to the way the spline data is treated. "Implicit on construction" refers to the construction process where topology and spline knot data is provided globally using a half-edge data structure and induced implicitly for each local and patch component. "Explicit on evaluation" refers to the evaluation process and the patchwork interpretation. As the information is implicitly induced upon construction, it is explicitly known on evaluation and can be baked into the construction during compilation, using a programming language with accommodating features. The principal construction process is visualized in Figure V.19. Therefore, the surface construction is well suited for parallelization. Additionally, changes to the topological structure can be considered locally (due to the minimal support property). The need for re-evaluation is limited to those of the blending spline patches affected by modifications to their local surfaces.

V.10.3.1 Interactive editing

The minimal support property makes the construction stable with respect to interactive editing due to the small data footprint generated by an issued update. This particular feature has been strongly advocated in the tensor-product variations of the construction. Even though a dynamic best-fit projection of the parametric polygons $\hat{\Omega}$ and $\bar{\Omega}$ have to be performed, it is also argued here. However, this can be optimized through pre-evaluation and/or fairing criteria with respect to global geometry artifacts [SV18]. Due to the intrinsic smoothing

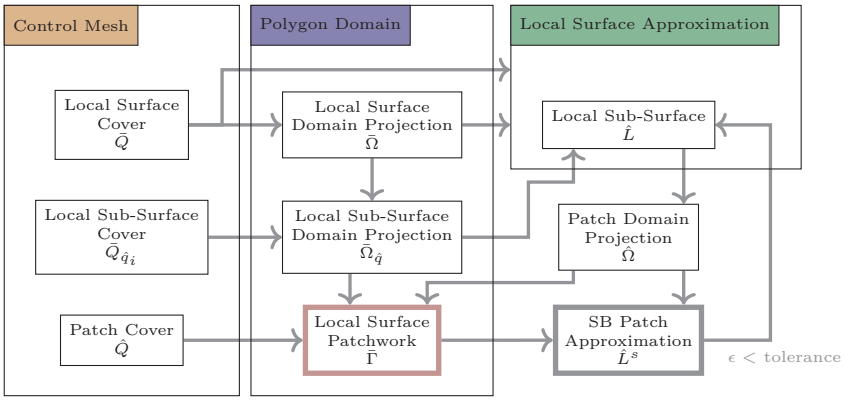


Figure V.19: Schematic visualization of an approximation process where a blending spline surface is to be fitted over some arbitrary topology data set.

properties of the blending spline construction, the geometric artifacts are expected to be less protruding. Furthermore, triggering of a best-fit parametric polygon recalculation can be controlled by considering a tolerance criterion.

V.10.3.2 Parallelization and GPU implementation

In [BDB15], tensor-product blending spline patch evaluation was developed for evaluation on modern graphical processing units and realized utilizing a modern GPU API, e.g., OpenGL 4.x [SA10] or DirectX 11 [Mic09], through the utilization of the tessellation shader and so-called patch-type primitives. These primitives can be interpreted as polygon domains of valence "2", 3, or 4 (iso-line, triangle- and tensor-product surface, respectively). Implementation of rendering kernels for poly-mesh blending spline patches of arbitrary valence requires the parametric polygon domain to be sub-partitioned to fit a configuration that matches the basic patch-type primitives. This is considered to be trivial as a mechanism depending on the structure and not the data. Additionally, the GPU graphics-specific evaluation technique can easily be extended to general-purpose APIs such as OpenCL [HM14; Mun14] or CUDA [Nvi14] and to new graphics APIs such as Vulkan [Tre15] or Metal [App21], as they support similar mechanisms.

Figure V.21 illustrates the surface equation, for a single patch, in the form of an evaluation process from left to right. Each horizontal chain can be thought of as a separate and independent computation pipeline. Hence, through its

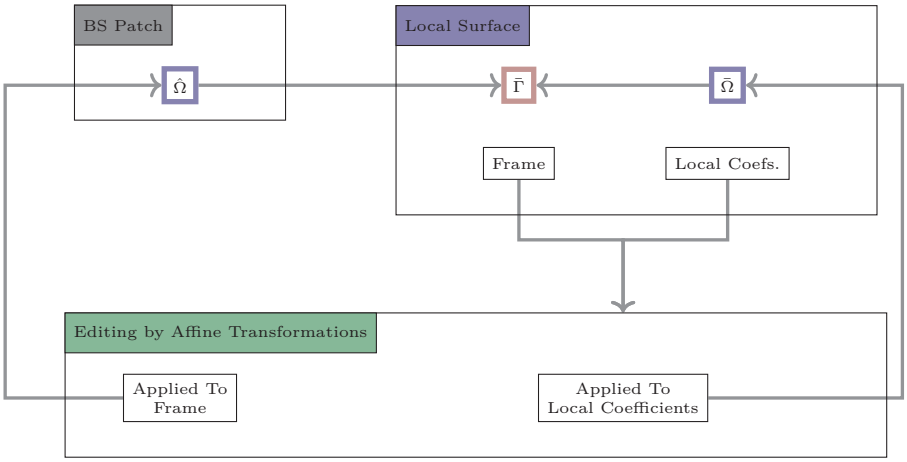


Figure V.20: Schematic visualization of the implications of an interactive modeling process where modifications, in the sense of affine transformations, are applied to either local surface coefficients or to the local surfaces themselves through frame transformations.

hierarchical nature, the construction can be parallelized on different levels. The vertical columns represent layers of potentially shared data, which can be cached for optimized evaluation. The color scheme corresponds to the colors used in Figure V.2. Implementation-wise, parallelization and layerwise data-sharing are to be considered as two extremes. Finding a good balance requires a trade-off performance analysis due to the construction's hierarchical nature.

The shared resources in question are mainly non-coefficient data or evaluations respective to the individual local surfaces and basis functions. Some of the shared resources could be pre-evaluated, typically for static resources, such as textures used for specifying material properties or colors. Other resources could use lazy-load intermediate buffers with level-of-detail access, typically when evaluating data from perspective views or spaces. Nevertheless, other resources could be evaluated directly, typically when memory access outside a kernel would be more expensive than direct evaluation. For instance, in multi-level systems: e.g., blending spline surfaces that also use blending spline surfaces as local surfaces. The choice of method depends on the application at hand and relevant trade-off analysis.

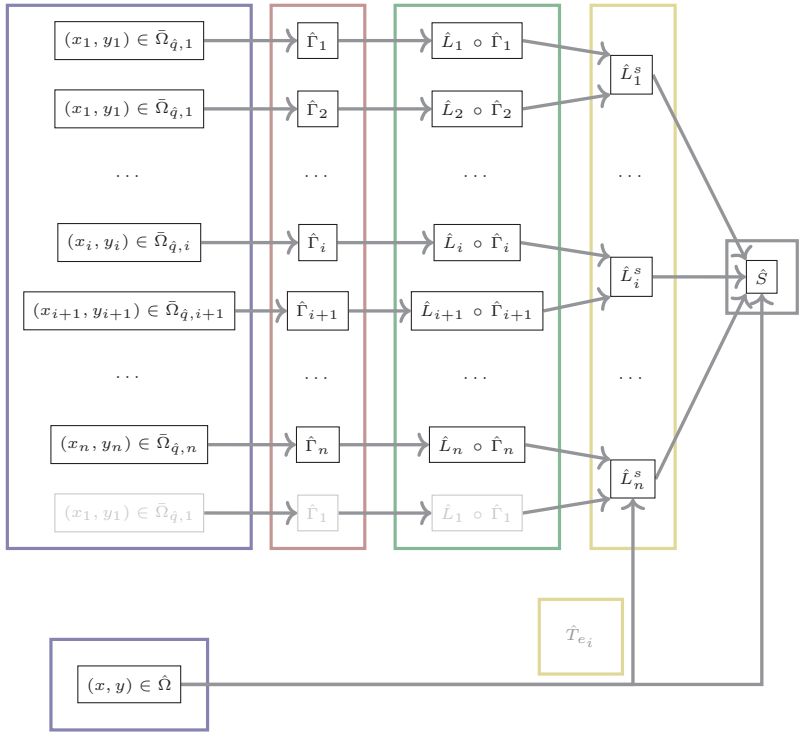


Figure V.21: The surface equation is visualized as an evaluation process.

V.10.3.3 Notes on tessellation

Parametric domain alignment is crucial for the evaluation process of blending spline patches. This needs to be handled due to the implicit nature of the polygon construction. All examples presented in this paper use a uniform planar tessellation, in a GBC sense, in order to not add any distortion to the geometry.

V.10.3.4 Notes on the development implementation

The current development implementation utilizes an in-house geometric modeling software suite written mainly in the C++17 programming language. Additionally, Blaze [Igl20] is utilized for vector- and matrix representations and computations, OpenMesh [Bot+02] for mesh loading and topology definitions, and the Qt libraries [The19] for application-, UX- and graphics-related features.

V.10.4 Future work

Given the complex nature of the construction and the focus of this article has been its description and initial smoothness control, a broader study into possible applications remains. Additionally, the following topics deserve priority in further investigations.

V.10.4.1 Efficient methods for interactive editing

The evaluation of a blending spline patch is an explicit process. However, the parametric polygon domain depends on the boundary of the embedded patch. Any modification to a local coefficient triggers a regeneration of the parametric polygon, as depicted in Figure V.19. Efficient solutions to this challenge are an interesting topic for future exploration.

V.10.4.2 Generalized barycentric coordinates

In Section V.6.4.1 it is stated that GBCs have not been in the primary scope of the current work. Using mean value coordinates in a setting where hierarchical blending and bijective mappings are essential turns out to be feasible but not optimal. It would be most interesting to investigate other GBC constructions in future explorations, both non-direct and directly evaluable variations, such as harmonic coordinates [DM06; Jos+07] or the power-based harmonic coordinates [Bud+16]. These variants could yield better results for domain control as they have strictly positive weights, even for non-convex domains.

References

[App21] Apple Inc. *Metal Developer Documentation*. 2021. URL: https://developer.apple.com/documentation/metal (cit. on p. 150).

[BZK09] Bommes, D., Zimmer, H., and Kobbelt, L. "Mixed-integer Quadrangulation". In: *ACM Transactions on Graphics* vol. 28, no. 3 (July 2009), 77:1–77:10 (cit. on p. 120).

[Bot+02] Botsch, M., Steinberg, S., Bischoff, S., and Kobbelt, L. "OpenMesh: A Generic and Efficient Polygon Mesh Data Structure". In: *OpenSG Symposium 2002*. 2002 (cit. on p. 152).

[BD21] Bratlie, J. and Dalmo, R. "Blending spline polygon surface over arbitrary poly-mesh topology". Under revision. (Cit. on p. 119).

[BDB15] Bratlie, J., Dalmo, R., and Bang, B. "Evaluation of smooth spline blending surfaces using GPU". In: *Curves and surfaces. 8th International Conference*. Ed. by Boissonnat, J.-D., Cohen, A., Gibaru, O., Gout, C., Lyche, T., Mazure, M.-L., and Schumaker, L. L. Vol. 9213. Lecture Notes in Computer Science. Springer, 2015, pp. 60–69 (cit. on pp. 123, 150).

[Bud+16] Budninskiy, M., Liu, B., Tong, Y., and Desbrun, M. "Power Coordinates: A Geometric Construction of Barycentrc Coordinates on Convex Polytopes". In: *ACM Transactions on Graphics* vol. 35, no. 6 (2016), p. 11 (cit. on p. 153).

[CHB09] Cottrell, J. A., Hughes, T. J. R., and Bazilevs, Y. *Isogeometric Analysis: Toward Integration of CAD and FEA*. 1st. Wiley Publishing, 2009 (cit. on p. 120).

[DBL09] Dechevsky, L. T., Bang, B., and Lakså, A. "Generalized Expo-Rational B-Splines". In: *International Journal of Pure and Applied Mathematics* vol. 57, no. 6 (2009), pp. 833–872 (cit. on p. 137).

[DLB06] Dechevsky, L. T., Lakså, A., and Bang, B. "Expo-Rational B-Splines". In: *International Journal of Pure and Applied Mathematics* vol. 27, no. 3 (2006), pp. 319–362 (cit. on p. 137).

[DM06] Derose, T. and Meyer, M. *Harmonic coordinates*. Pixar Technical Memo 06-02. Pixar Animation Studios, 2006 (cit. on p. 153).

[DLP13] Dokken, T., Lyche, T., and Pettersen, K. F. "Polynomial splines over locally refined box-partitions". In: *Computer Aided Geometric Design* vol. 30, no. 3 (2013), pp. 331–356 (cit. on p. 121).

[Far02] Farin, G. *Curves and surfaces for CAGD: a practical guide*. 5th. Academic Press, 2002 (cit. on pp. 121, 123).

[Flo15] Floater, M. S. "Generalized barycentric coordinates and applications". In: *Acta Numerica* vol. 24 (2015), pp. 161–214 (cit. on pp. 121, 125, 126).

[GJS12] Giannelli, C., Jüttler, B., and Speleers, H. "THB-splines: The truncated basis for hierarchical splines". In: *Computer Aided Geometric Design* vol. 29, no. 7 (2012). Geometric Modeling and Processing 2012, pp. 485–498 (cit. on p. 121).

[HK18] Hettinga, G. J. and Kosinka, J. "Multisided generalisations of Gregory patches". In: *Computer Aided Geometric Design* vol. 62 (2018), pp. 166–180 (cit. on p. 121).

[HF06] Hormann, K. and Floater, M. S. "Mean Value Coordinates for Arbitrary Planar Polygons". In: *ACM Transactions on Graphics* vol. 25, no. 4 (2006), pp. 1424–1441 (cit. on p. 126).

[HM14] Howes, L. and Munshi, A. *The OpenCL API Specification (Version 2.0)*. 2014. URL: http://khronos.org/opencl (cit. on p. 150).

[Igl20] Iglberger, K. *Blaze C++ Linear Algebra Library*. (Bitbucket). Aug. 2020. URL: http://bitbucket.org/blaze-lib (cit. on p. 152).

[Jos+07] Joshi, P., Meyer, M., DeRose, T., Green, B., and Sanocki, T. "Harmonic Coordinates for Character Articulation". In: *ACM Transactions on Graphics* vol. 26, no. 3 (July 2007) (cit. on p. 153).

[Ket99] Kettner, L. "Using generic programming for desgning a data structure for polyhedral surfaces". In: *Computational Geometry* vol. 13, no. 1 (1999), pp. 65–90 (cit. on p. 123).

[Kob00] Kobbelt, L. P. "Discrete fairing and variational subdivision for freeform surface design". In: *The Visual Computer* vol. 16, no. 3 (May 2000), pp. 142–158 (cit. on p. 140).

[LS07] Lai, M.-J. and Schumaker, L. L. *Spline Functions on Triangulations*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2007 (cit. on p. 120).

[Lak07] Lakså, A. "Basic properties of Expo-Rational B-splines and practical use in Computer Aided Geometric Design". (Dr.philos.) PhD thesis. University of Oslo, 2007 (cit. on pp. 122, 142, 149).

[Lak13] Lakså, A. "ERBS-surface construction on irregular grids". In: *39th International conference applications of mathematics in engineering and economics AMEE13*. Ed. by Pasheva, V. and Venkov, G. Vol. 1570. AIP Conference Proceedings. AIP Publishing, 2013, pp. 113–120 (cit. on p. 122).

[LB15] Lakså, A. and Bang, B. "Surface construction on irregular grids". In: *Large-Scale Scientific Computing 2015*. Ed. by Lirkov, I., Margenov, S., and Waśniewski, J. Vol. 9374. Lecture Notes in Computer Science. Springer, 2015, pp. 385–393 (cit. on p. 133).

[LD89] Loop, C. T. and DeRose, T. D. "A Multisided Generalization of Bézier Surfaces". In: *ACM Transactions on Graphics* vol. 8, no. 3 (1989), pp. 204–234 (cit. on pp. 121, 138).

[Mic09] Microsoft® corporation. *Direct3D 11 Features*. 2009. URL: http://docs.microsoft.com/en-us/windows/win32/direct3d11/direct3d-11-features (cit. on p. 150).

[Mun14] Munshi, A. *The OpenCL C Language Specification (Version 2.0)*. 2014. URL: http://khronos.org/opencl (cit. on p. 150).

[Nvi14] Nvidia. *CUDA C Programming guide v.6.5*. 2014. URL: http://developer.nvidia.com/cuda-toolkit-65 (cit. on p. 150).

[PBP02] Prautzsch, H., Boehm, W., and Paluszny, M. *Bézier and B-spline Techniques*. Mathematics and visualization. Berlin Heidelberg: Springer-Verlag, 2002 (cit. on p. 123).

[SV18] Salvi, P. and Varady, T. "Multi-sided Bezier surfaces over concave polygonal domains". In: *Computers & Graphics* vol. 74 (2018), pp. 56–65 (cit. on pp. 121, 126, 138, 143, 149).

[SVR14] Salvi, P., Várady, T., and Rockwood, A. "Ribbon-based transfinite surfaces". In: *Computer Aided Geometric Design* vol. 31, no. 9 (2014), pp. 613–630 (cit. on p. 140).

[Sch17] Schneider, T. "Theory and Applications of Bijective Barycentric Mappings". PhD thesis. Universita della Svizzera, 2017 (cit. on p. 126).

[SS06] Schumaker, L. L. and Sorokina, T. "Smooth Macro-Elements on Powell-Sabin-12 Splits". In: *Mathematics of Computation* vol. 75, no. 254 (2006), pp. 711–726 (cit. on p. 120).

[SW12] Schumaker, L. L. and Wang, L. "Splines on Triangulations with Hanging Vertices". In: *Constructive Approximation* vol. 36, no. 3 (Dec. 2012), pp. 487–511 (cit. on p. 120).

[Sed+03] Sederberg, T. W., Zheng, J., Bakenov, A., and Nasri, A. "T-splines and T-NURCCs". In: *ACM SIGGRAPH 2003 Papers*. SIGGRAPH '03. San Diego, California: ACM, 2003, pp. 477–484 (cit. on pp. 121, 133).

[SA10] Segal, M. and Akeley, K. *The OpenGL Graphics System: A Specification (Version 4.0 (Core Profile))*. Mar. 2010. URL: http://khronos.org/registry/OpenGL/specs/gl/glspec40.core.pdf (cit. on p. 150).

[The19] The Qt Company. *Qt - Complete software development framework*. 2019. URL: http://qt.io (cit. on p. 152).

[Tre15] Trevett, N. *Vulkan Press Briefing*. Press Briefing by the Khronos group at SIGGRAPH 2015. 2015. URL: http://khronos.org/vulkan (cit. on p. 150).

[VRS11] Várady, T., Rockwood, A., and Salvi, P. "Transfinite surface interpolation over irregular n-sided domains". In: *Computer Aided Design* vol. 43, no. 11 (2011). Solid and Physical Modeling 2011, pp. 1330–1340 (cit. on pp. 121, 140).

[VSK16] Várady, T., Salvi, P., and Karikó, G. “A Multi-sided Bézier Patch with a Simple Control Structure”. In: *Computer Graphics Forum* vol. 35, no. 2 (2016), pp. 307–317 (cit. on pp. 121, 138).

[VSK17] Várady, T., Salvi, P., and Kovács, I. “Enhancement of a multi-sided Bézier surface representation”. In: *Computer Aided Geometric Design* vol. 55, no. Supplement C (2017), pp. 69–83 (cit. on pp. 121, 126, 138).

Appendices

Appendix A

Exploring future C++ features: source code

The following appendix contains the source code listing of the principal implementation API discussed in Paper IV. The source code is written for GCC with C++20 support as of November 2019. The API is a compilable proof of concept and is to be treated as a template.

A.1 Basic types

Listing A.1 defines basic vector and matrix types for use with an affine space representation.

Listing A.1: *basic_types.h*

```
#ifndef GM_BASIC_TYPES_H
#define GM_BASIC_TYPES_H

// stl
#include <array>


namespace basic_types
{

  template <typename Unit_T, size_t Dim_T>
  using Vector = std::array<Unit_T, Dim_T>;

  template <typename Unit_T, size_t Dim_T>
  using Matrix = std::array<Vector<Unit_T, Dim_T>, Dim_T + 1>;
}   // namespace basic_types

#endif   // GM_BASIC_TYPES_H
```

A.2 Spaces

Listing A.2 defines an affine space and an affine space object type. The affine space type contains type-information only and is intended for use as a compile-time meta-object only.

Listing A.2: *space.h*

```
#ifndef GM_SPACE_H
#define GM_SPACE_H

#include "basic_types.h"


namespace space
{
  template <typename Unit_T, size_t Dim_T>
  struct AffineSpace {
    static constexpr auto Dim = Dim_T;
    using Unit                = Unit_T;
    using Point               = basic_types::Vector<Unit, Dim>;
    using Vector              = basic_types::Vector<Unit, Dim>;
    using Frame               = basic_types::Matrix<Unit, Dim>;
  };

  template <typename EmbedSpace_T>
  struct AffineSpaceObject {
    using EmbedSpace = EmbedSpace_T;

    static constexpr auto Dim = EmbedSpace_T::Dim;
    using Unit                = typename EmbedSpace_T::Unit;
    using Point               = typename EmbedSpace_T::Point;
    using Vector              = typename EmbedSpace_T::Vector;
    using Frame               = typename EmbedSpace_T::Frame;

    Frame frame = {/* identity */};

    void translate(Vector)
    { /* math */
    }
  };
}   // namespace space

#endif   // GM_SPACE_H
```

A.3 Parametrics

Listing A.3 defines two implementation helper classes for parametric objects and sub-objects. These are intended to be combined with using-statements.

Listing A.3: *parametric.h*

```
#ifndef GM_PARAMETRIC_H
#define GM_PARAMETRIC_H

#include "space.h"

// stl
#include <cmath>

namespace parametric
{

  template <typename Kernel_T>
  struct ParametricObjectImpl : Kernel_T {

    // Context
    using Base   = Kernel_T;
    using Kernel = Kernel_T;

    // Embed Space
    using EmbedSpaceObject    = typename Kernel::EmbedSpaceObject;
    using EmbedSpace          = typename EmbedSpaceObject::EmbedSpace;
    static constexpr auto Dim = EmbedSpace::Dim;
    using Unit                = typename EmbedSpace::Unit;
    using Point               = typename EmbedSpace::Point;
    using Vector              = typename EmbedSpace::Vector;
    using Frame               = typename EmbedSpace::Frame;

    // Parametric space
    using PSpace                    = typename Kernel::PSpace;
    static constexpr auto PSpaceDim = PSpace::Dim;
    using PSpaceUnit                = typename PSpace::Unit;
    using PSpacePoint               = typename PSpace::Point;
    using PSpaceVector              = typename PSpace::Vector;
    using PSpaceFrame               = typename PSpace::Frame;

    // Transitive constructor
    template <typename... Ts>
    explicit ParametricObjectImpl(Ts&&... ts)
      : Base(std::forward<Ts>(ts)...)
    {
    }
  };


  template <typename Kernel_T>
  struct PSubPSpaceObjectImpl : ParametricObjectImpl<Kernel_T> {

    // Context
    using Base             = ParametricObjectImpl<Kernel_T>;
    using Kernel           = Kernel_T;
```

```
51      using PSpaceObject     = typename Kernel::PSpaceObject;
52      using ParametricObject = typename Kernel::ParametricObject;
53
54      // PSpace == PSpaceObject PSpace
55      using PSpace                    = typename Kernel::PSpace;
56      static constexpr auto PSpaceDim = PSpace::Dim;
57      using PSpaceUnit                = typename PSpace::Unit;
58      using PSpacePoint               = typename PSpace::Point;
59      using PSpaceVector              = typename PSpace::Vector;
60      using PSpaceBoolArray           = std::array<bool, PSpaceDim>;
61      using PSpaceSizeArray           = std::array<size_t, PSpaceDim>;
62
63      // Embed Space == ParametricObject Embed Space
64      using EmbedSpaceObject    = typename Kernel::EmbedSpaceObject;
65      using EmbedSpace          = typename EmbedSpaceObject::EmbedSpace;
66      static constexpr auto Dim = EmbedSpace::Dim;
67      using Unit                = typename EmbedSpace::Unit;
68      using Point               = typename EmbedSpace::Point;
69      using Vector              = typename EmbedSpace::Vector;
70
71      // ParametricObject PSpace == PSpaceObject Embed Space
72      using ParametricObject_PSpace =
73        typename Kernel::ParametricObject_PSpace;
74      static constexpr auto ParametricObject_PSpaceDim
75        = ParametricObject_PSpace::Dim;
76      using ParametricObject_PSpaceUnit =
77        typename ParametricObject_PSpace::Unit;
78      using ParametricObject_PSpacePoint =
79        typename ParametricObject_PSpace::Point;
80      using ParametricObject_PSpaceVector =
81        typename ParametricObject_PSpace::Vector;
82
83
84      // Delete default contructor
85      PSubPSpaceObjectImpl() = delete;
86
87      // Transitive constructor
88      template <typename... Ts>
89      explicit PSubPSpaceObjectImpl(ParametricObject* obj, Ts&&... ts)
90        : Base(obj, std::forward<Ts>(ts)...)
91      {
92      }
93    };
94
95
96  }   // namespace parametric
97
98
99
100 #endif   // GM_PARAMETRIC_H
```

A.4 Parametric kernels

Listing A.4 define three boiler-plate example kernels for three different use cases; a circle(1D), a torus(2D), and a circle embedded in the parameter space of a torus (sub-object, e.g., 1D in 2D).

Listing A.4: *parametric_kernels.h*

```
#ifndef GM_PARAMETRIC_KERNELS_H
#define GM_PARAMETRIC_KERNELS_H

#include "space.h"

// stl
#include <cmath>



#define GM_PARAMETRIC_KERNEL_TYPES                                          \
  /* Parametric Space */                                                    \
  static constexpr auto PSpaceDim = PSpace::Dim;                            \
  using PSpaceUnit                = typename PSpace::Unit;                  \
  using PSpacePoint               = typename PSpace::Point;                 \
  using PSpaceVector              = typename PSpace::Vector;                \
  using PSpaceBoolArray           = std::array<bool, PSpaceDim>;            \
  using PSpaceSizeArray           = std::array<size_t, PSpaceDim>;          \
                                                                            \
  /* Embed Space */                                                         \
  using EmbedSpaceObject    = EmbedSpaceObject_T;                           \
  using EmbedSpace          = typename EmbedSpaceObject::EmbedSpace;        \
  static constexpr auto Dim = EmbedSpace::Dim;                              \
  using Unit                = typename EmbedSpace::Unit;                    \
  using Point               = typename EmbedSpace::Point;                   \
  using Vector              = typename EmbedSpace::Vector;



namespace parametric::kernel
{

  template <typename EmbedSpaceObject_T>
  struct CircleKernel : EmbedSpaceObject_T {

    // Context
    using Base = EmbedSpaceObject_T;

    // Affine Space: 1D
    using PSpace = space::AffineSpace<typename Base::Unit, 1ul>;

    // Inherited Kernel types
    GM_PARAMETRIC_KERNEL_TYPES

    // Construction
    template <typename... Ts>
    CircleKernel(Unit radius = 3, Ts&&... ts)
```

```
49        : Base(std::forward<Ts>(ts)...), r{radius}
50      {
51      }
52
53      // Members
54      Unit r;
55
56      // Polymorphic functionality
57      PSpacePoint     startParameter() const { return {0}; }
58      PSpacePoint     endParameter() const { return {2 * M_PI}; }
59      PSpaceBoolArray isClosed() const { return {true}; }
60
61      using Result = Point;
62      template <typename = void>
63      Result evaluate(PSpacePoint par) const
64      {
65        const auto& [t] = par;
66
67        const auto x = r * std::cos(t);
68        const auto y = r * std::sin(t);
69
70        if constexpr (Dim == 2)
71          return Point{x, y};   // 2D
72        else if constexpr (Dim == 3)
73          return Point{x, y, 0};   // 3D
74      }
75    };
76
77
78    template <typename EmbedSpaceObject_T>
79    struct TorusKernel : EmbedSpaceObject_T {
80
81      // Context
82      using Base = EmbedSpaceObject_T;
83
84      // Parametric Space: 2D
85      using PSpace = space::AffineSpace<typename Base::Unit, 2ul>;
86
87      // Inherited Kernel types
88      GM_PARAMETRIC_KERNEL_TYPES
89
90      // Construction
91      template <typename... Ts>
92      explicit TorusKernel(Unit wheelrad = 3, Unit tuberad1 = 1,
93                           Unit tuberad2 = 1, Ts&&... ts)
94        : Base(std::forward<Ts>(ts)...), wr{wheelrad}, tr1{tuberad1},
95          tr2{tuberad2}
96      {
97      }
98
99      // Members
100     Unit wr;
101     Unit tr1;
102     Unit tr2;
103
104
105     /////
```

```
106     // Polymorphic functionality
107     PSpacePoint startParameter() const { return {0, 0}; }
108     PSpacePoint endParameter() const { return {2 * M_PI, 2 * M_PI}; }
109     PSpaceBoolArray isClosed() const { return {true, true}; }
110
111     using Result = Point;
112     Result evaluate(PSpacePoint par) const
113     {
114       const auto& [u, v] = par;
115
116       const auto x = std::cos(u) * (tr1 * std::sin(v) + wr);
117       const auto y = std::sin(u) * (tr1 * std::sin(v) + wr);
118       const auto z = std::sin(v) * tr2;
119
120       // Embeded in 3D projective space
121       if constexpr (Dim == 3) return Point{x, y, z};
122     }
123   };
124
125
126   template <typename PSpaceObject_T, typename ParametricObject_T,
127             typename EmbedSpaceObject_T>
128   struct CurveInSurfaceKernel : EmbedSpaceObject_T {
129
130     // Context
131     using Base             = EmbedSpaceObject_T;
132     using PSpaceObject     = PSpaceObject_T;
133     using ParametricObject = ParametricObject_T;
134
135     // PSpace == PSpaceObject PSpace
136     using PSpace                    = typename PSpaceObject::PSpace;
137
138
139     // Inherited Kernel types
140     GM_PARAMETRIC_KERNEL_TYPES
141
142     // ParametricObject PSpace == PSpaceObject Embed Space
143     using ParametricObject_PSpace = typename ParametricObject::PSpace;
144     static constexpr auto ParametricObject_PSpaceDim
145       = ParametricObject_PSpace::Dim;
146     using ParametricObject_PSpaceUnit =
147       typename ParametricObject_PSpace::Unit;
148     using ParametricObject_PSpacePoint =
149       typename ParametricObject_PSpace::Point;
150     using ParametricObject_PSpaceVector =
151       typename ParametricObject_PSpace::Vector;
152
153
154     // Construction
155     template <typename... Ts>
156     explicit CurveInSurfaceKernel(ParametricObject* obj, Ts&&... ts)
157       : Base(),
158         pspace_object(std::forward<Ts>(ts)...), parametric_object{obj}
159     {
160     }
161
162
```

```
163     // Members
164     PSpaceObject      pspace_object;
165     ParametricObject* parametric_object;
166
167
168     /////
169     // Polymorphic functionality
170
171     PSpacePoint startParameter() const
172     {
173       return pspace_object.startParameter();
174     }
175     PSpacePoint endParameter() const
176     {
177       return pspace_object.endParameter();
178     }
179     PSpaceBoolArray isClosed() const
180     {
181       return pspace_object.isClosed();
182     }
183
184     using Result = Point;
185     Result evaluate(PSpacePoint par) const
186     {
187       const auto pspace_res = pspace_object.evaluate(par);
188       const auto parametric_object_res
189         = parametric_object->evaluate(pspace_res);
190
191       return parametric_object_res;
192     }
193   };
194
195 }   // namespace parametrics::kernel
196
197
198
199
200 #endif   // GM_PARAMETRIC_KERNELS_H
```

A.5 Geometric modeling API

The API uses the parametric helper classes combined with the kernels to define a cleaner and user-friendly API (defined in Listing A.5).

Listing A.5: *api.h*

```
#ifndef GM_API_H
#define GM_API_H

#include "parametric_kernels.h"
#include "parametric.h"

namespace geomod
{


  template <template <typename> typename Kernel_T,
            typename EmbedSpaceObject_T>
  using ParametricObject
    = parametric::ParametricObjectImpl<Kernel_T<EmbedSpaceObject_T>>;

  template <template <typename> typename PSpaceKernel_T,
            template <typename, typename, typename> typename Kernel_T,
            typename ParametricObject_T>
  using ParametricSubObject = parametric::PSubPSpaceObjectImpl<
    Kernel_T<parametric::ParametricObjectImpl<
               PSpaceKernel_T<space::AffineSpaceObject<
                 typename ParametricObject_T::PSpace>>>,
             ParametricObject_T,
             space::AffineSpaceObject<
               typename ParametricObject_T::EmbedSpace>>>;


  using ProjSpace    = space::AffineSpace<double, 3ul>;
  using ProjSpaceObj = space::AffineSpaceObject<ProjSpace>;

  using Circle
    = ParametricObject<parametric::kernel::CircleKernel, ProjSpaceObj>;

  using Torus
    = ParametricObject<parametric::kernel::TorusKernel, ProjSpaceObj>;

  using SubCircleInTorus
    = ParametricSubObject<parametric::kernel::CircleKernel,
                          parametric::kernel::CurveInSurfaceKernel,
                          Torus>;

}   // namespace geomod


#endif   // GM_API_H
```

A.6 Usage

Listing A.6 defines a couple of API usage examples.

Listing A.6: *main.cpp*

```
1
2  #include "api.h"
3
4
5  int main(int args, char** argv)
6  {
7
8    auto circle          = geomod::Circle{};
9    auto torus           = geomod::Torus{};
10   auto torus_subcircle = geomod::SubCircleInTorus{&torus, 2.0};
11
12   const auto circle_eval
13     = circle.evaluate(geomod::Circle::PSpacePoint{});
14
15   const auto torus_eval
16     = torus.evaluate(geomod::Torus::PSpacePoint{});
17
18   const auto torus_subcircle_eval = torus_subcircle.evaluate(
19     geomod::SubCircleInTorus::PSpacePoint{});
20
21   return 0;
22 }
```

A.7 Building

Listing A.7 defines a basic CMake build file.

Listing A.7: *CMakeLists.txt*

```
1  cmake_minimum_required(VERSION 3.11)
2
3  project(modern_cpp_geomod_api VERSION 0.1 LANGUAGES CXX)
4
5  set( SRCS main.cpp)
6
7  add_executable( ${PROJECT_NAME})
8  target_sources( ${PROJECT_NAME} PRIVATE ${SRCS})
9
10 target_compile_features( ${PROJECT_NAME}
11   PRIVATE $<$<CXX_COMPILER_ID:GNU>:cxx_std_20>)
12
13 target_compile_options( ${PROJECT_NAME}
14   PRIVATE $<$<CXX_COMPILER_ID:GNU>: -pedantic -Wall -Werror -Wno-unused-but-
       set-variable>)
15
16 set_target_properties( ${PROJECT_NAME}
17   PROPERTIES CXX_EXTENSIONS OFF)
```

Appendix B

GMlib source code examples

The following appendix contains C++ example code listings referenced in the introductory sections of the thesis. To run these examples, download and compile a copy of GMlib2 and the template demo application from their respective source code repositories [UiT21]. A current setup guide can be found on the wiki pages associated with the template demo source code repository. Information valid as of: May 14, 2021. The author will also be able to provide a copy of the required source code in the future, upon request – probably ^^,
(The source code is licensed under *The Unlicense*).

B.1 Supplementary parametric sub-object examples

Related examples, referenced from the thesis, showing how to construct and define geometric objects. Visual variations of these examples are shipped with the GMlib2 template demo.

B.1.1 Sub-curve in torus

GMlib2 code example for the construction in Section 4.3.

```
namespace gm2  = gmlib2;
namespace gm2p = gm2::parametric;

using ProjSpaceObj = gm2::ProjectiveSpaceObject<>;
using SubCurveType = gm2p::SubCurveInSurface<gm2p::HermiteCurveP2V2,
                                             Torus, ProjSpaceObj>;
using SCPoint      = SubCurveType::PSpaceObject_Point;
using SCVector     = SubCurveType::PSpaceObject_Vector;

// Torus
auto torus = make_unique<Torus>(3.0, 1.0, 1.0);

// SubCurve on torus
auto const subcurve_p0 = SCPoint{0.0, 0.0};
auto const subcurve_p1 = SCPoint{2 * M_PI, 1.5 * M_PI};
auto const subcurve_v0 = SCVector{20.0, 0.0};
auto const subcurve_v1 = SCVector{2.0, 0.0};
auto       subcurve    = make_unique<SubCurveType>(
  torus.get(), subcurve_p0, subcurve_p1, subcurve_v0, subcurve_v1);
```

B.1.2 Sub-curve in surface in surface

GMlib2 code example for the construction in Section 4.3.1.

```
namespace gm2  = gmlib2;
namespace gm2p = gm2::parametric;

using ProjSpaceObj = gm2::ProjectiveSpaceObject<>;
using SubPlaneType
  = gm2p::SubSurfaceInSurface<gm2p::Plane, Torus, ProjSpaceObj>;
using SSPoint  = SubPlaneType::PSpaceObject_Point;
using SSVector = SubPlaneType::PSpaceObject_Vector;

using SubLineType
  = gm2p::SubCurveInSurface<gm2p::Line, SubPlaneType, ProjSpaceObj>;
using SCPoint  = SubLineType::PSpaceObject_Point;
using SCVector = SubLineType::PSpaceObject_Vector;

// Torus
auto torus = make_unique<Torus>(3.0, 1.0, 1.0);

// SubSurface on torus
auto const subsurface_p = SSPoint{0.0, 0.0};
auto const subsurface_u = SSVector{0.4, 0.0};
auto const subsurface_v = SSVector{0.0, M_PI};
auto subsurface = make_unique<SubPlaneType>(torus.get(), subsurface_p,
                                            subsurface_u, subsurface_v);

// SubCurve on subsurface
auto const subcurve_p = SCPoint{0.2, 0.2};
auto const subcurve_v = SCVector{0.6, 0.6};
auto       subcurve
  = make_unique<SubLineType>(subsurface.get(), subcurve_p, subcurve_v);
```

B.1.3 Sub-surface in volume

GMlib2 code example for the construction in Section 4.3.2.

```
namespace gm2  = gmlib2;
namespace gm2p = gm2::parametric;

using ProjSpaceObj = gm2::ProjectiveSpaceObject<>;
using BezierVolume = gm2p::BezierVolume<ProjSpaceObj>;
using ControlNet   = BezierVolume::ControlNet;
using Point        = BezierVolume::Point;
using Unit         = BezierVolume::Unit;

ControlNet C;
/* Fill C */

auto bezvolume = make_unique<BezierVolume>(C);

using SubTorusType
  = gm2p::SubSurfaceInVolume<gm2p::Torus, BezierVolume, ProjSpaceObj>;

auto subtorus
  = make_unique<SubTorusType>(bezvolume.get(),
                              /*wheel radius*/, /*inner tube radius*/,
                              /*outer tube radius*/);
```

B.1.4 Sub-surface in polygon

GMlib2 code example for the construction in Section 4.3.3.

```
namespace gm2  = gmlib2;
namespace gm2p = gm2::parametric;
namespace gbp  = gm2p::generalizedbezierpatch;

using ProjSpaceObj = gmlib2::ProjectiveSpaceObject<>;
using GBPatch      = gm2p::GeneralizedBezierPatch<ProjSpaceObj>;
using ControlNets  = GBPatch::ControlNets;
using Point        = GBPatch::Point;

using SubPolygon
  = gm2p::SubPolygonSurfaceInPolygonSurface<gm2p::Polygon, GBPatch,
                                            ProjSpaceObj>;
using SPPoint    = SubPolygon::PSpaceObject_Point;
using SPVertices = SubPolygon::PSpaceObject::Vertices;


// Polygon
const auto P = gm2::DVectorT<Point>{
  Point{0.0, 0.0, 0.0}, Point{2.0, 0.0, 0.0}, Point{4.0, 0.0, 2.0},
  Point{4.0, 0.0, 4.0}, Point{0.0, 0.0, 2.0}};

// Ribbon surface control nets
auto gbpoly55_cn_pi   = gbp::constructControlNet<GBPatch>(P, 5, 0);
auto gbpoly55_cn_pip1 = gbp::constructControlNet<GBPatch>(P, 5, 1);
auto gbpoly55_cn_pip2 = gbp::constructControlNet<GBPatch>(P, 5, 2);
auto gbpoly55_cn_pip3 = gbp::constructControlNet<GBPatch>(P, 5, 3);
auto gbpoly55_cn_pip4 = gbp::constructControlNet<GBPatch>(P, 5, 4);

// --Displace--
gbpoly55_cn_pi(2, 2)[1]   = 5.0;
gbpoly55_cn_pip2(2, 1)[1] = 0.5;
gbpoly55_cn_pip4(1, 1)[1] = 4.0;

// Generalized Bezier patch
auto gbpoly5 = make_unique<GBPatch>(
  ControlNets{gbpoly55_cn_pi, gbpoly55_cn_pip1, gbpoly55_cn_pip2,
              gbpoly55_cn_pip3, gbpoly55_cn_pip4},
  scenario.root());

// Sub-polygon polygon surface
SPVertices sub_vertices{SPPoint{-0.5, -0.5}, SPPoint{0.5, -0.5},
                        SPPoint{0.5, 0.5}, SPPoint{-0.5, 0.5}};
auto subpolygon = make_unique<SubPolygon>(gbpoly5.get(), sub_vertices);
```

B.2 Supplementary differential operator examples

This section provides an example that demonstrates how to construct an operator for computation of directional derivatives of embedded sub-objects. The construction uses the deducible compile-time meta information to determine static object types and values such as spatial dimensions and value and object types. Variations of these examples can be found in GMlib2 [UiT21].

B.2.1 Directional derivative

A directional derivative operator which we can utilize in the evaluator of a sub-object can be defined as follows. The operator takes the embedded and the embedding objects as parameters together with a position and parameter direction. Additionally, it has two finite difference operators, which it uses to compute the directional derivatives. One for the embedded object, `pspace_op`, and one for the embedding object, `op`. When computing the directional derivative, we first must compute the direction in the space of the embedded object (lines 16-18) before computing the directional derivative in the embedding object's own space (lines 19-23). Notice that the latter operator is defined and utilized for the embedding object. Its relationship to a parent space is expected to be handled on the outside of this function.

```
1  template <typename FiniteDifferenceOperator_T
2            = divideddifference::fd::CentralDifference>
3  struct SubObjectDirectionalDerivativeLocal {
4
5    template <typename PSpaceObject_T, typename ParametricObject_T>
6    auto operator()(
7      const PSpaceObject_T& pspace_obj,
8      const typename std::decay_t<PSpaceObject_T>::PSpacePoint&  pos,
9      const typename std::decay_t<PSpaceObject_T>::PSpaceVector& dir,
10     const ParametricObject_T& pobj) const
11   {
12     using PSpaceObject      = decay_t<PSpaceObject_T>;
13     using ParametricObject  = decay_t<ParametricObject_T>;
14     using PSpaceObjEvalCtrl = typename PSpaceObject::PObjEvalCtrl;
15
16     const auto DpspaceH = pspace_op(pspace_obj, pos, dir);
17     const auto Dpspace
18       = subvector<0UL, ParametricObject::PSpaceVectorDim>(DpspaceH);
19     const auto Dp
20       = op(pobj,
21            PSpaceObjEvalCtrl::toPosition(pspace_obj.evaluateParent(
22              pos, typename PSpaceObject::PSpaceSizeArray())),
23            normalize(Dpspace) * length(dir));
24     return pair(Dp, length(Dpspace));
25   }
26
27   static constexpr DirectionalDerivativeParent<
28     FiniteDifferenceOperator_T>
29     pspace_op{};
30
31   static constexpr DirectionalDerivativeLocal<
32     FiniteDifferenceOperator_T>
33     op{};
34 };
```

We continue and define the two operators for computing directional derivatives. The first computes the directional derivative, given a parametric position, a derivative, and the given finite difference operator.

```
template <typename FiniteDifferenceOperator_T
          = divideddifference::fd::CentralDifference>
struct DirectionalDerivativeLocal {

  template <typename ParametricObject_T>
  auto
  operator()(const ParametricObject_T&                        pobj,
             const typename ParametricObject_T::PSpacePoint&  pos,
             const typename ParametricObject_T::PSpaceVector& dir) const
  {
    using PObjEvaluationCtrl =
      typename ParametricObject_T::PObjEvalCtrl;
    const auto fn = [&pobj](const auto& par) {
      const auto res = pobj.evaluateLocal(par);
      return evaluate(PObjEvaluationCtrl::toPosition(res));
    };

    auto res = typename ParametricObject_T::VectorH();
    subvector<0UL, ParametricObject_T::VectorDim>(res)
      = op(pos, dir, fn);
    res[ParametricObject_T::VectorDim] =
      typename ParametricObject_T::Unit(0);
    return res;
  }

  static constexpr FiniteDifferenceOperator_T op{};
};
```

The second operator utilizes the evaluation from the first and maps it into an object-related “parent space”.

```
template <typename FiniteDifferenceOperator_T
          = divideddifference::fd::CentralDifference>
struct DirectionalDerivativeParent
  : DirectionalDerivativeLocal<FiniteDifferenceOperator_T> {
private:
  using Base = DirectionalDerivativeLocal<FiniteDifferenceOperator_T>;

public:
  template <typename ParametricObject_T>
  auto operator()(
    const ParametricObject_T&                                pobj,
    const typename decay_t<ParametricObject_T>::PSpacePoint&  pos,
    const typename decay_t<ParametricObject_T>::PSpaceVector& dir) const
  {
    return pobj.pSpaceFrameParent() * Base::operator()(pobj, pos, dir);
  }
};
```

References

[UiT21] UiT - The Arctic University of Norway. *GMlib/GMlib2 C++ Geometric modeling library*. (NeIC). May 2021. URL: http://source.coderefinery.org/gmlib (cit. on pp. 171, 175).

Appendix C

Additional poly-mesh images

C.1 Blending spline polygon surface basis functions

For the blending spline polygon patch, we utilizes two special basis functions based on the basis functions in [VRS11]. The basis functions are visualized in Figure C.1. The basis function illustrated on the bottom row is utilized to blend along the side parameter s, see (V.6), while the basis function illustrated on the top row is utilized to blend along the side parameter h, see (V.7).

C.2 Blending spline polygon surface examples

Figure C.2 shows a blending spline polygon surface approximated from eight vertices tessellated into a poly-mesh topology and where the center vertex is elevated. Each local surface is a 4^{th}-degree GB-patch, and interpolates the vertices at their respective natural associated interpolation point. The resulting surface is visualized with isophotes, internal overlapping boundary-edge normals, and with wireframe. The isophotes and the overlapping patchwise boundary-edge normals primarily display continuity. However, while the isophote rendering can visually argue G^2-smoothness, the normals show how the internal "linear" (parametric $[x, y]$-plane) boundaries sway in the embedded space. Additionally, overlapping patchwise boundary-edge normals means that each patch renders their normals around their individual boundaries, and the normals on adjacent boundaries are rendered "twice". The wireframe rendering indicates how the parameter space is distributed in the embedded space, as it is sampled from a regularized triangulation.

Figure C.3 shows an approximation of the same data set as in Figure C.2. However, all vertices now lie in the plane, and one of the boundary vertices is edited afterward. It shows three versions, two illustrated with isophotes and one with wireframe. As with Figure C.2, the wireframe illustration shows the distribution of the parametric spaces, patchwise. The isophote versions demonstrate the intrinsic minimal support property of the blending spline construction. In the middle figure one of the local patches is rendered transparently. Additionally, some of the local coefficients have been translated.

Figure C.4 shows the same approximation as in Figure C.3. However, the local surface associated with the center vertex has been translated vertically and rotated about 90-degrees in the horizontal plane. The center local surface has been edited through vertical translation. The isophote rendering illustrates continuity, and the solid color shading illustrates the blending spline patch partitions.

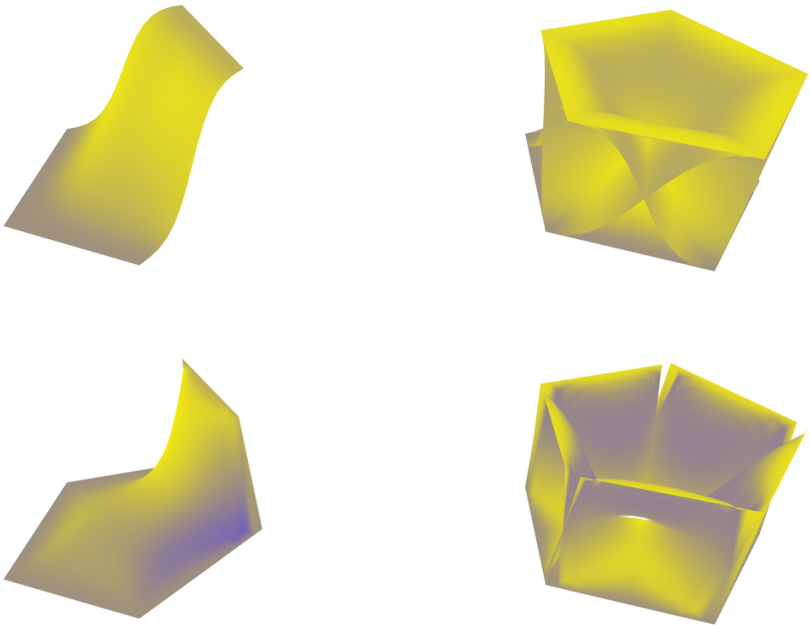


Figure C.1: Gooch-rendered visualizations of the basis functions used in the process of blending local polygon sub-surfaces to construct the blending spline polygon patch. Top: h-direction blending bases, (V.7). Bottom: s-direction blending bases, (V.6).

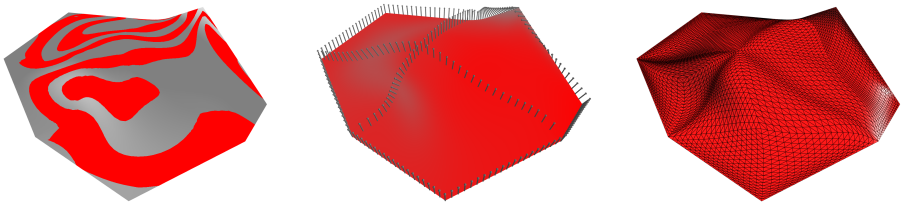


Figure C.2: Blending spline polygon surface approximation of a poly-mesh topology consisting of seven boundary- and one internal vertex. The poly-mesh is tessellated into polygons of three, four ($\times 2$) and five sides. Left-to-right rendering: isophotes, overlapping patchwise boundary-edge normals and patchwise wireframes.

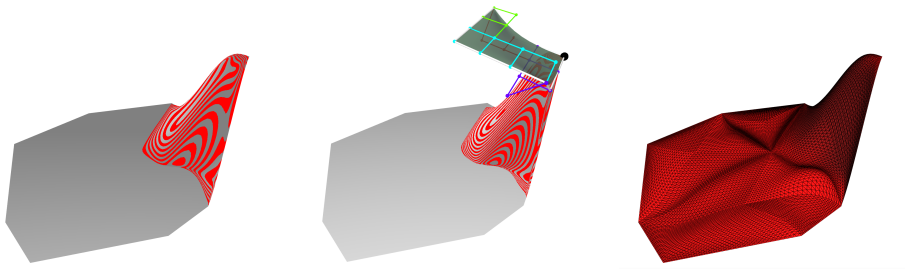


Figure C.3: Blending spline polygon surface approximation of a poly-mesh topology consisting of seven boundary- and one internal vertex. The poly-mesh is tessellated into polygons of three, four (×2), and five sides. The right-most local boundary surface has been edited: transformation applied to both the local surface itself and the local surfaces' own local coefficients. Left-to-right rendering: isophotes, isophotes w/ local surface, and patchwise wireframes.

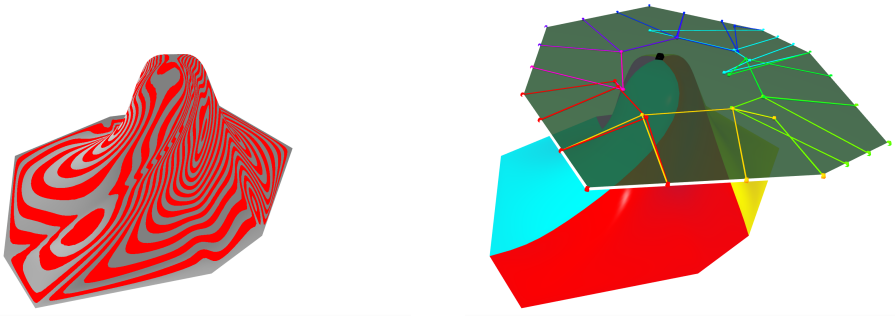


Figure C.4: Blending spline polygon surface approximation of a poly-mesh topology consisting of seven boundary- and one internal vertex. The poly-mesh is tessellated into polygons of three, four (×2), and five sides. Left-to-right rendering: isophotes and solid color rendering w/ local surface.

References

[VRS11] Várady, T., Rockwood, A., and Salvi, P. "Transfinite surface interpolation over irregular n-sided domains". In: *Computer Aided Design* vol. 43, no. 11 (2011). Solid and Physical Modeling 2011, pp. 1330–1340 (cit. on p. 179).