Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 316 (2017) 880–943
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

Post-processing and visualization techniques for isogeometric analysis results

Annette Stahl[a,*], Trond Kvamsdal[a], Christian Schellewald[b]

[a] Department of Mathematical Sciences, NTNU - Norwegian University of Science and Technology, Alfred Getz' vei 1, 7491 Trondheim, Norway
[b] Department of Computer and Information Science, NTNU - Norwegian University of Science and Technology, Sem Sælands vei 7-9, 7491 Trondheim, Norway

Available online 8 November 2016

Abstract

Isogeometric Analysis (IGA) introduced in 2005 by Hughes et al. (2005) [1] exploits one mathematical basis representation for computer aided design (CAD), geometry and analysis during the entire engineering process. In this paper we extend this concept also for visualization. The presented post-processing and visualization techniques thereby strengthen the relation between geometry, analysis and visualization. This is achieved by facilitating the same mathematical function space used for geometry and analysis also for post-processing and visualization purposes.

During non-linear analysis derivatives are incrementally computed and stored with different basis function representations. We introduce and investigate projection methods to be able to use the same function space for both displacements and stresses without loss of accuracy.

To obtain a common representation for structured and unstructured meshes like hierarchical spline, locally refined B-spline (LR B-spline) and T-spline techniques we exploit Bézier decomposition in a post-processing step resulting in a Bézier element representation and constitute it as generalized representation. The typically used unrelated (fictitious) finite element mesh representation for visualization purposes are easily replaced without changing the underlying geometry as well as the algorithmic data structure. One further benefit of the used Bézier decomposition lies in the fact that it facilitates a natural parallel implementation on Graphics Processor Units (GPUs) exploiting shader programming.

In this paper we have developed and investigated an *accurate, efficient and practical post-processing pipeline for visualization of isogeometric analysis results*. The proposed IGA visualization pipeline consists of three steps: (1) *Projection*, (2) *Bézier decomposition* and (3) *Pixel-accurate rendering*. We have tested four different projection methods. A description on how to perform Bézier decomposition of LR B-splines are given (whereas for hierarchical and T-splines this has been done before). Furthermore, the use of GPU shader programming to enable efficient and pixel-accurate visualization is detailed.

The performance of the four different projection techniques has been tested on manufactured problems as well as on realistic benchmark problems. Furthermore, the IGA visualization pipeline has been demonstrated on a number of real-world applications.

* Correspondence to: Department of Engineering Cybernetics, NTNU - Norwegian University of Science and Technology, O.S. Bragstads plass 2D, NO-7034 Trondheim, Norway.
E-mail addresses: annette.stahl@ntnu.no (A. Stahl), trond.kvamsdal@ntnu.no (T. Kvamsdal), christian.schellewald@ntnu.no (C. Schellewald).

http://dx.doi.org/10.1016/j.cma.2016.10.040

Keywords: Visualization; Isogeometric analysis; Finite element analysis; Bézier decomposition; Graphics processor unit; Locally refined B-splines

1. Introduction

Scientific visualization understood as a method to represent visually scientific results based on multi-dimensional large scale data sets is of significant importance in various fields, especially in computational mechanics and engineering. Today, many visualization tools and instruments are available to serve special needs in each particular field, without building the connection to the underlying mathematical modeling concept for design and analysis.

The relatively new idea of isogeometric analysis [1] allows the unification of the mathematical modeling concept behind engineering design, computational analysis and simulation. The same mathematical model representation for CAD, geometry modeling and analysis generated from Non-Uniform Rational B-Splines (NURBS) can be exploited to construct a single geometric model which is used during the entire engineering process.

In order to extend the isogeometric picture, we propose to facilitate the same mathematical function space used for design and analysis also for the mathematical modeling of the visualization process (cf. Fig. 1 right). We strengthen the relation between geometry, analysis and visualization — resulting in high quality visualizations of computational results in a natural way.

Material behavior, like deformation and failure of solid structures or fluid mechanical phenomena are described through solutions of appropriate Partial Differential Equations (PDEs). In state-of-the-art methodologies Finite Element Methods (FEMs) are used in order to find approximate solutions of PDEs.

During conventional Finite Element Analysis (FEA) every processing step (CAD model design, geometry modeling with meshing and visualization) employs a different mathematical representation (cf. Fig. 1 left). CAD models are generated using a standard representation based on NURBS. For analysis a different description based on Lagrange elements is used, that approximates the geometry. In a further post-processing step, mathematically unrelated and approximately linear finite element meshes are used to visualize the computed results. As a consequence, we have to deal with the storage and data structure of different representations that makes the simulation process computational expensive, time consuming and less accurate.

In order to present visualization results of higher quality new visualization techniques have to be employed to preserve smoothness on complex and higher-order geometries. Especially, when it comes to arbitrary large scale geometric topologies represented by high-order basis functions a guaranteed accuracy is desired. For such visualizations with an arbitrary level of detail local refinement techniques like LR B-Splines [2,3], T-Splines [4,5] and hierarchical splines [6,7] can be exploited.

We address these issues by providing a platform for generalization by exploiting spline properties and techniques like Bézier decomposition to decompose structured and unstructured mesh representations into a Bézier element representation [8–10]. With respect to efficiency and accuracy, this structure can easily be incorporated into IGA visualization pipelines allowing for a parallel computing architecture and simplifying the implementation structure for local refinement technologies.

Furthermore, we focus on the challenge to embed the visualization of second-order representations like strain and stress. Both physical entities describing material behavior and failure in structural mechanics and engineering. Strains and stresses are projected during post-processing into the same spline function space in isogeometric analysis as the displacement allowing for an efficient data representation for the visualization process. As the spline basis in general is not interpolating nodal values we have to resort to other methods than simple nodal averaging like in traditional FEA. We derive methods that take full advantage of the increased regularity of the underlying spline function space.

Typically, Lagrange polynomials are the primal unknown in FEA formulations, i.e. the displacements are represented as weighted sum of the displacements in the finite element nodes:

$$u(x) = \sum_{i=1}^{n} \phi_i(x) u_i . \tag{1}$$

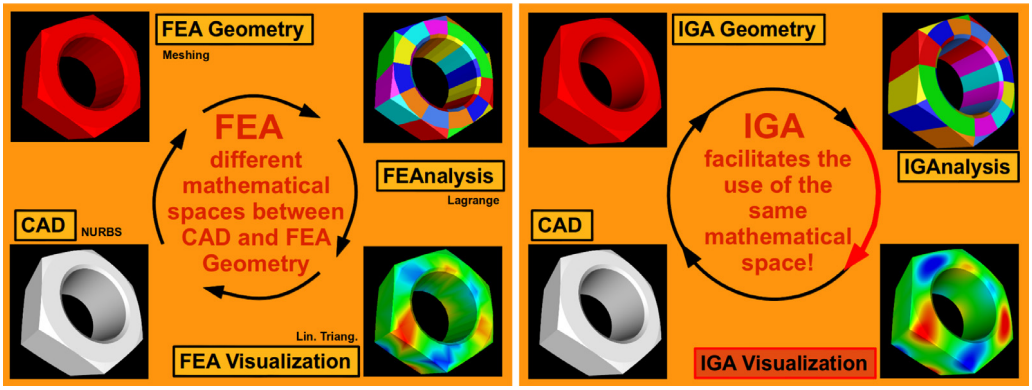

Fig. 1. **Left:** Conventional Finite Element Analysis (FEA) pipeline. Typically, different representations are used for CAD design, geometry and visualization. **Right:** Extended Isogeometric Analysis (IGA) pipeline. The same mathematical function space is facilitated during the entire engineering process — from design to visualization.

Here $u(x)$ is the displacement at the location x, ϕ_i is the Lagrange nodal basis function for node i and u_i the displacement in node i. The finite dimensional displacement space $X = \{\phi_i\}_{i=1}^n$ is typical C^0-continuous and interpolating the nodal values. Thus, for visualization of the displacement it is enough to know the polynomial order of the basis functions and the displacement information for the nodes. This may be stored as further attribute at the nodes and is easily interpreted within the visualization pipeline.

If we look at strain and stress representations for the linear and second-order elements in 1D we see that the strains ϵ, are the gradient of the displacement:

$$\epsilon = \frac{du}{dx}. \tag{2}$$

These piecewise constant and linear functions are discontinuous in the (element end) nodes and represent a challenge for the visualization step as more than one value in the nodes have to be stored. Furthermore, the interpolation function for the strains implicate the derivatives of the basis functions ϕ_i. This problem has traditionally been handled in classical finite element programs by *projecting* strain and stresses into the same function space as the displacement. E.g. for a linear element in 1D one takes the average of the strains in the nodes and interpolates these values using the basis functions ϕ_i for the displacements.

The strains ϵ for linear problems in 2D are given by the infinitesimal strain tensor determined by the following kinematical relation

$$\epsilon = \frac{1}{2}(\nabla + \nabla^\top)u, \tag{3}$$

where ∇ is the usual 2D gradient operator. The relation between strains and stresses is given by

$$\sigma = C \cdot \epsilon, \tag{4}$$

where C is the fourth-order constitutive tensor. This linear relationship is well known as Hooke's law [11]. In 2D the component stresses are defined in Cartesian components of the Cauchy second-order symmetric stress tensor

$$\sigma = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy} \end{bmatrix}, \tag{5}$$

where $\sigma_{xy} = \sigma_{yx}$. In order to display such spatial varying stresses using for example a contour plot we may display each component separately. However, an alternative is to compute a scalar quantity that represents the stress tensor, like the von Mises stress $\bar{\sigma}$ that in the 2D Cartesian coordinate system is defined as follows

$$\bar{\sigma} = \sqrt{\sigma_{xx}^2 - \sigma_{xx}\sigma_{yy} + \sigma_{yy}^2 + 3\sigma_{xy}^2}. \tag{6}$$

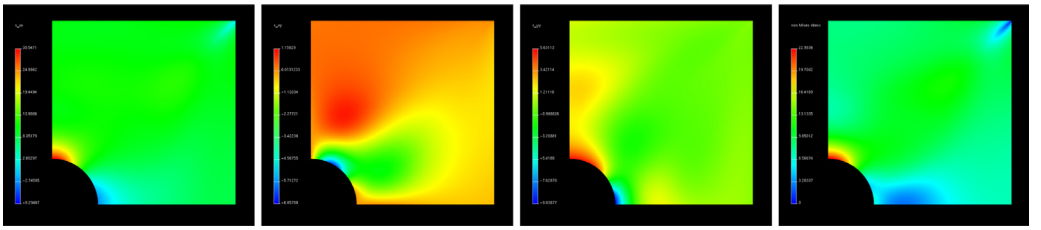

Fig. 2. 2D circular hole problem: Stress tensor components σ_{xx}, σ_{xy}, σ_{yy} and von Mises stress as defined in (5) and (6) respectively.

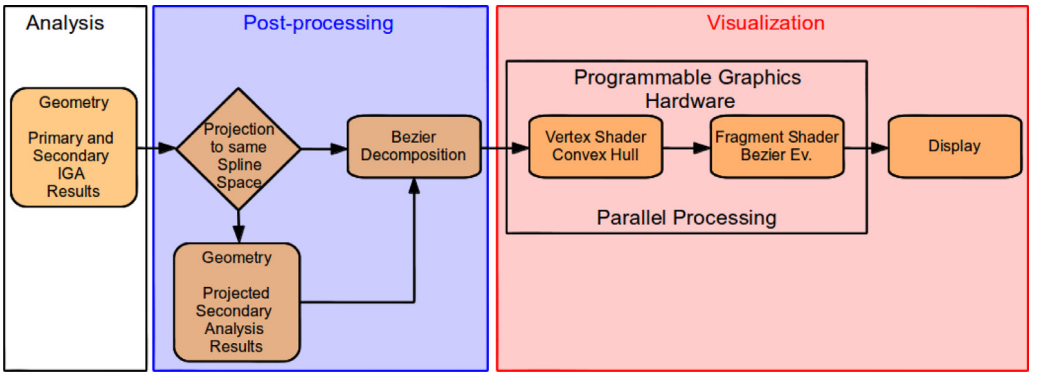

Fig. 3. IGA visualization pipeline: Analysis, Post-processing and Visualization of IGA results. During post-processing a projection of secondary analysis results into the primary spline function space is performed as well as a Bézier decomposition. Visualization: Parallel processing is directly performed on the GPU.

Fig. 2 shows contour plots representing the scalar field values for the three different stress tensor components and the von Mises stress for the well-known 2D circular hole problem.

For 2D and 3D problems either nodal averaging or more elaborated projection schemes like local or global L_2-projections can be used to allow for the use of just one basis function set during the visualization of the results. We see that for results like von Mises there will always be a need for some kind of interpolation/projection as the function variation is non-polynomial.

Generally, for non-linear problems derivatives have to be computed incrementally during analysis. As a consequence, usually different basis function representations have to be stored. We emphasize that for the in this paper presented approach only one basis function set has to be stored. In the following section we provide an overview of the proposed IGA visualization pipeline.

1.1. IGA visualization pipeline

The IGA visualization pipeline can be divided into three parts: First the *Analysis* where the results one wishes to show are computed, secondly the *Post-processing* involving the projection of the results followed by a Bézier decomposition and thirdly the *Visualization* itself which is based on a GPU implementation. These pipeline steps are described in more detail below and an illustration of it with post-processing of the analysis results (projection of secondary IGA results and Bézier decomposition) is depicted in Fig. 3.

Analysis: Primary and secondary results, like respectively displacements and stresses are generated by IGA software tools.

Post-processing — Projection: In case of second-order representations secondary analysis results are projected into the same spline function space as the primary results. Having all result data within the same spline function space only one basis function representation for the visualization process has to be stored creating a convenient and efficient data structure. The particular projection methods we exploit are explained in more detail in Section 2.

Post-processing — Bézier decomposition: In order to provide a uniform data structure suitable (generalized) for structured and unstructured mesh handling – as well as to facilitate parallel processing on GPU – spline patches are

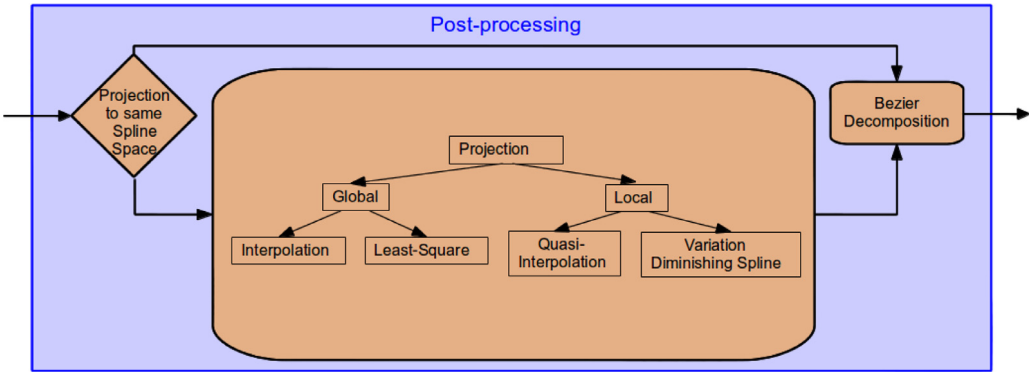

Fig. 4. Post-processing: Efficient data handling by exploiting projection methods.

subdivided using Bézier decomposition techniques. These are discussed in Section 3 with respect to the underlying mesh representations (like unstructured local refined meshes as LR B-splines (cf. Section 3.2), T-splines and hierarchical splines).

Visualization — Display: Geometry as well as primary and (when required) projected secondary IGA results are stored in form of image-textures. Shader programs are generated depending on the polynomial degree of the geometry data. Vertex primitives (including knot vectors, control points, etc.) are provided to the vertex shader where – in parallel – for each Bézier sub-patch the convex hull of its control points is computed. This is used within the later executed ray-casting algorithm (explained in Section 4.2.4) in the fragment program, where a per-pixel Bézier value evaluation is performed. At this stage also the color and the shading of the object are computed which leads to the final graphical output on the display. More details can be found in Section 4.

2. Post-processing: Projection of secondary quantities

During analysis primary (e.g displacement) and secondary (e.g derivatives) results are incrementally computed and stored with different basis function representations. In this section we derive global and local projection methods to represent for example strain and stresses within the displacement function space, so that only one basis function set needs to be stored (cf. Fig. 4) for visualization purposes. We have chosen to study in detail four different projection methods some of them are commonly used for splines, to see if they are suitable in the context of isogeometric finite element analysis. We are not aware of that any such in depth studies have been pursued before and therefore provide valuable new insight which projection techniques are preferable in the IGA visualization pipeline context.

2.1. Preliminaries

Established approximation and interpolation methods are defined and compared with each other in Appendix A.2 in order to provide the mathematical terms to construct higher-order B-Spline and NURBS curves, surfaces and volumes which fit arbitrary second-order functions. A more detailed introduction and overview of underlying geometric concepts and algorithms can be found in [12–14].

In the following we derive new projection methods based on the approximation and interpolation techniques introduced in Appendix A.2.

2.2. Global projection methods

A given 1D displacement field (i.e. a scalar field) is defined by spline basis functions as follows:

$$u(\xi) = \sum_{i=1}^{n} N_{ip}(\xi) u_i. \tag{7}$$

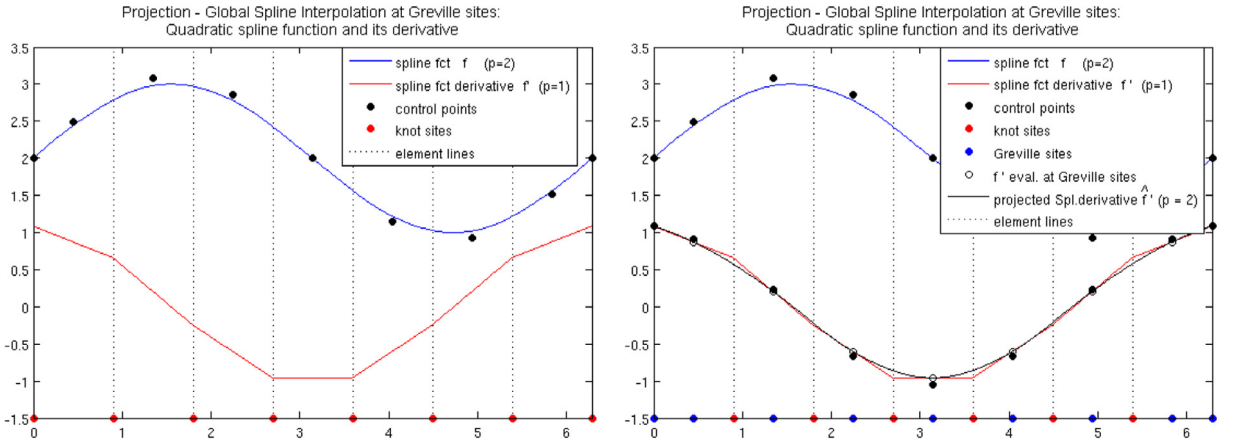

Fig. 5. Projection using global spline interpolation at Greville points: **Left:** Quadratic spline function $f \in \mathbb{S}_{2,\Xi}$ (blue) and its linear derivative $f' \in \mathbb{S}_{1,\Xi}$ (red). **Right:** Greville points are determined using Eq. (71) illustrated as blue dots; spline derivative $f' \in \mathbb{S}_{1,\Xi}$ evaluated at Greville points (clear dots) as interpolation points; obtained control point set (black dots) as solution of system (9); projected spline derivative $\hat{f}' \in \mathbb{S}_{2,\Xi}$ (black) representing a quadratic spline (cf. (10)). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

In 1D the corresponding strains are defined as the derivative of the displacement field with respect to the spatial variable x. By using the chain rule we get

$$\begin{aligned}\frac{\partial u}{\partial x}(\xi) &= \frac{\partial u}{\partial \xi}\frac{\partial \xi}{\partial x} \\ &= \frac{\partial u}{\partial \xi}\left(\frac{\partial x}{\partial \xi}\right)^{-1} \\ &= \frac{\sum_{i=1}^{n} N'_{ip}(\xi)u_i}{\sum_{i=1}^{n} N'_{ip}(\xi)x_i}.\end{aligned} \tag{8}$$

Assuming that the Jacobian of the mapping between ξ and x coordinates are constant the denominator above is a constant independent of ξ. The strains are thus given by the derivatives of the basis function of the displacements.

In the following we will modify global methods, like spline interpolation at Greville points and least-squares approximation as well as local methods, like variation diminishing spline approximation and quasi-interpolation as presented in Appendix A.2. Exploiting these methods we create spline derivative functions, representing stress quantities belonging to the same spline function space as the displacement quantity. This form of representing a function in one spline space into another spline space is called a projection.

2.2.1. *Spline interpolation exploiting Greville points*

We start by computing the derivative $f'(\xi)$ of a spline function $f(\xi)$ with respect to given control weights c and knot vector Ξ (cf. Fig. 5 left). The spline function f itself belongs to the spline space $\mathbb{S}_{p,\Xi}$, while the derivative f' of one degree lower belongs to the spline space $\mathbb{S}_{p-1,\Xi}$. In order to project the derivative f' into the same spline space as the spline function f, we seek appropriate control function values. First we evaluate the derivative at the Greville points ξ^* (cf. Eq. (71)) obtain $f'(\xi^*)$ and solve as in (75) a linear system of equations

$$Ac^* = \begin{pmatrix} N_{1,p}(\xi_1^*) & \cdots & N_{n,p}(\xi_1^*) \\ \vdots & \ddots & \vdots \\ N_{1,p}(\xi_n^*) & \cdots & N_{n,p}(\xi_n^*) \end{pmatrix} \begin{pmatrix} c_1^* \\ \vdots \\ c_n^* \end{pmatrix} = \begin{pmatrix} f'(\xi_1^*) \\ \vdots \\ f'(\xi_n^*) \end{pmatrix} = f'(\xi^*), \tag{9}$$

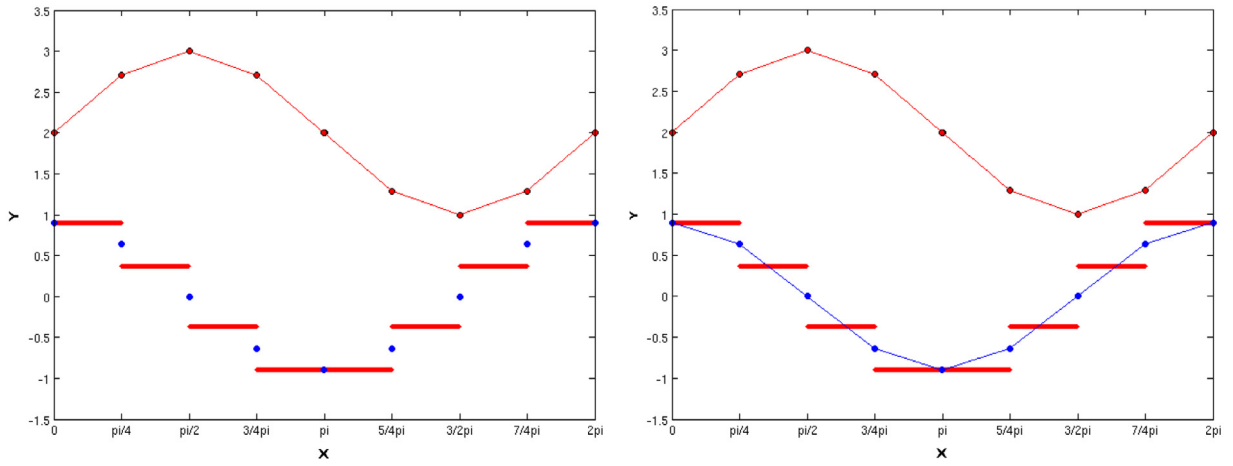

Fig. 6. Projection: C^0-continuity case — general idea. **Left:** Linear spline function and its dis-continuous derivative with Greville points. **Right:** Spline derivative evaluated at Greville points used to generate a linear spline derivative representation.

where $0 < A \in \mathbb{S}_{p,\Xi}$ with diagonal bandwidth at least $p+1$. We obtain as solution a new set of control points c^*. This new control point set is used to design a derivative

$$\hat{f}'(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi) c_i^* \quad \in \mathbb{S}_{p,\Xi} \tag{10}$$

belonging to the same spline space $\mathbb{S}_{p,\Xi}$ as the spline function f (cf. Fig. 5 right).

C^0-continuity case. A special case for this projection technique occurs when C^0-continuity in the derivative function is approached as illustrated in Fig. 6. The Greville points developed for the derivative function are not evaluated at the left or right sites instead the averaging is performed between the left and right evaluation points to generate a midpoint evaluation as shown in Fig. 6.

2.2.2. *Least-squares approximation*

Exploiting the least-squares approximation method (Appendix Eq. (76) following) we design a second global projection method. In this case the spline derivative function $f'(\xi) \in \mathbb{S}_{p-1,\Xi}$ of a spline function $f(\xi) \in \mathbb{S}_{p,\Xi}$ with respect to given control weights c and knot vector Ξ are evaluated at so called defined Gaussian points following the Gaussian quadrature rules (discussed in [15]) depending on n_I integration points.

According to the polynomial degree of the spline space $S_{p,\Xi}$ of the spline function f and the respective knot vector Ξ we compute for each knot span $\xi_i < \xi_i + 1$ (element) Gaussian points g_j and weights w_j, $j = 1 : n_I$.

We evaluate the derivative $f' \in S_{p-1,\Xi}$ at the integration points g_j and define b as follows

$$b := \begin{pmatrix} f'(g_1)\sqrt{w_1} \\ \vdots \\ f'(g_{n_I(n-p)})\sqrt{w_{n_I(n-p)}} \end{pmatrix}. \tag{11}$$

Using (81) the mass matrix is computed and we solve the following system

$$A^\top A\, c^* = A^\top b, \tag{12}$$

where the coefficient matrix $N = A^\top A$ is symmetric and positive semi-definite, with

$$A = \begin{pmatrix} N_{1,p}(g_1)\sqrt{w_1} & \cdots & N_{n,p}(g_1)\sqrt{w_1} \\ \vdots & \ddots & \vdots \\ N_{1,p}(g_{n_I(n-p)})\sqrt{w_{n_I(n-p)}} & \cdots & N_{n,p}(g_{n_I(n-p)})\sqrt{w_{n_I(n-p)}} \end{pmatrix}. \tag{13}$$

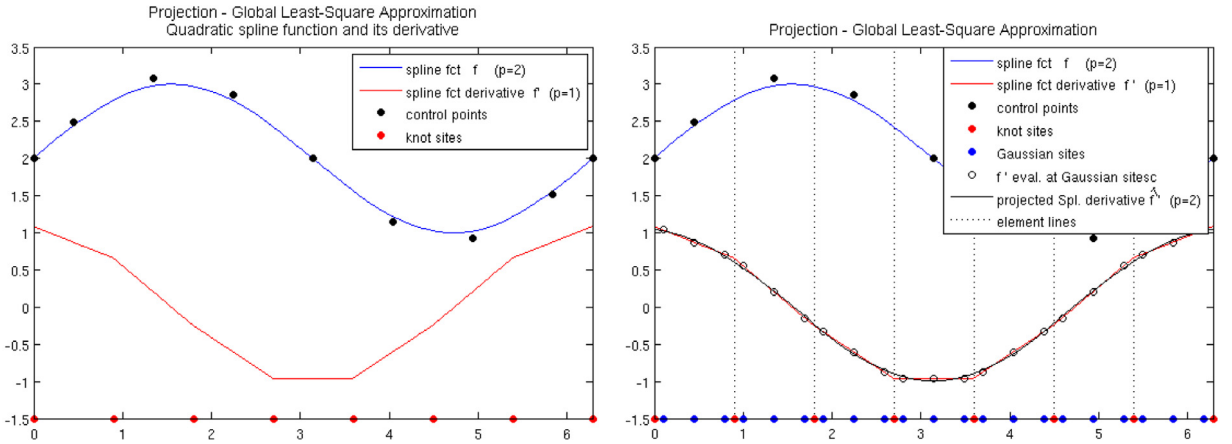

Fig. 7. Projection using global least-squares approximation method: **Left:** Quadratic spline function $f \in \mathbb{S}_{2,\Xi}$ (blue) and its linear derivative $f' \in \mathbb{S}_{1,\Xi}$ (red). **Right:** Gaussian sites are determined using the Gaussian quadrature rule for $n_I = 3$ illustrated as blue dots; spline derivative $f' \in \mathbb{S}_{1,\Xi}$ evaluated at Gauss points (clear dots); computed control point set (black dots); projected spline derivative $\hat{f}' \in \mathbb{S}_{2,\Xi}$ (black) representing a quadratic spline. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The matrix A is positive definite and therefore non-singular and the solution is unique if and only if A has linearly independent columns. The $n \times n$-matrix $N = A^\top A$ is symmetric

$$\|A\|_2^2 \geq 0. \tag{14}$$

The solution c^* of Eq. (12) is used as control point set to design a projected spline derivative

$$\hat{f}'(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi) c_i^* \quad \in \mathbb{S}_{p,\Xi}. \tag{15}$$

An illustrative example is shown in Fig. 7.

2.3. Local projection methods

2.3.1. Variation diminishing spline approximation

One of the simplest and very useful spline approximation methods is the Variation Diminishing Spline Approximation (VDSA) described in the Appendix Eq. (83). We make use of this method and evaluate the spline derivative $f' \in S_{p-1,\Xi}$ at the previous stated Greville points ξ^* and obtain $f'(\xi^*)$. To obtain the variation diminishing approximation Vb to the derivative, we simply use the function values $f'(\xi^*)$ as B-spline coefficients $b_j^* = f_j'(\xi^*)$ directly and obtain

$$\hat{f}'(\xi) = \sum_{j=1}^{n} N_{j,p}(x) f_j'(\xi^*), \tag{16}$$

$$(Vb)(\xi) = \sum_{j=1}^{n} N_{j,p}(x) b_j^*, \tag{17}$$

where the approximation Vb belongs to the spline space $\mathbb{S}_{p,\Xi}$. This method is a generalization of piecewise linear interpolation and has a shape preserving behavior (cf. Fig. 8).

2.3.2. Quasi-interpolation

Restricting the interpolation method using Greville points from Section 2.2.1 on an interval and exploiting the quasi-interpolation recipe as described in Appendix A.2.2 Quasi-interpolation we are able to construct a quasi-interpolant $\hat{f}'(\xi) \in S_{p,\Xi}$ of a spline function $f(\xi) \in S_{p,\Xi}$ as follows:

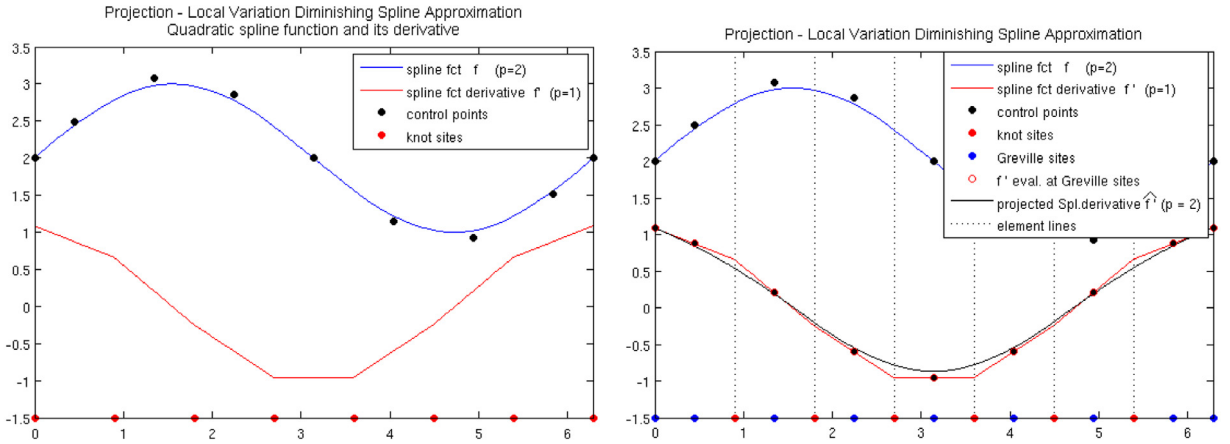

Fig. 8. Projection using local variation diminishing spline approximation: **Left:** Quadratic spline function $f \in \mathbb{S}_{2,\Xi}$ (blue) and its linear derivative $f' \in \mathbb{S}_{1,\Xi}$ (red). **Right:** Greville points are computed illustrated as blue dots; spline derivative $f' \in \mathbb{S}_{1,\Xi}$ evaluated at Greville points (clear red dots); evaluated spline derivative at Greville points used as control point set (black dots); projected spline derivative $\hat{f}' \in \mathbb{S}_{2,\Xi}$ (black) representing a quadratic spline. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

In order to compute the control points c_j locally as described in Section 2.2.1 the local support property of splines has to be considered. Starting from a spline space of degree p for a function f the support of the spline N_j is defined on the interval $[\xi_j, \xi_{j+p+1}]$.

Choosing the interval $I = [\xi_{j+1}, \xi_{j+p}]$ the local spline space $S_{p,\xi,I}$ has dimension $2p-1$ and is spanned by the B-splines $\{N_i\}_{i=j-(p-1)}^{i=j+(p-1)}$.

We use interpolation as a local approximation method P^I. The quasi-interpolant should reproduce the whole spline space and we use therefore P^I to reproduce $S_{p,\xi,I}$.

In order to retrieve a unique solution for the system of linear equations we need a B-spline coefficient matrix during the interpolation process which is non-singular, that is the case if and only if its diagonal is positive. As we know the dimension of the spline space $S_{p,\xi,I}$ is $2p-1$ we need $2p-1$ interpolation points. We use the p knots $\xi_{j+1}, \ldots, \xi_{j+p}$ and one point from each of the knot intervals in I, such that for $k = 1, 2, \ldots, 2p-1$

$$x_{j,k} = \begin{cases} \xi_{j+(k+1)/2} & \text{if } k \text{ is odd,} \\ \dfrac{\xi_{j+k/2} + \xi_{j+k/2+1}}{2} & \text{if } k \text{ is even.} \end{cases} \tag{18}$$

To determine the solution of the local interpolation problem $P^I f$ we have to solve the linear system of $2p-1$ equations in the $2p-1$ unknowns $c_{j-(p-1)}, \ldots, c_{j+(p-1)}$ given by

$$P^I f'(x_{j,k}) = \sum_{i=j-(p-1)}^{j+(p-1)} N_{i,p}(x_{j,k}) c_i = f'(x_{j,k}), \quad k = 1, 2, \ldots, 2p-1. \tag{19}$$

In matrix–vector form this becomes

$$A_j c_j = \begin{pmatrix} N_{j-(p-1),p}(x_{j,1}) & \cdots & N_{j+(p-1),p}(x_{j,1}) \\ \vdots & \ddots & \vdots \\ N_{j-(p-1),p}(x_{j,2p-1}) & \cdots & N_{j+(p-1),p}(x_{j,2p-1}) \end{pmatrix} \begin{pmatrix} c_{j-(p-1)} \\ \vdots \\ c_{j+(p-1)} \end{pmatrix} = \begin{pmatrix} f'(x_{j,1}) \\ \vdots \\ f'(x_{j,2p-1}) \end{pmatrix} = f'_j(x). \tag{20}$$

Because of the way we have chosen the interpolation points all the entries on the diagonal of the coefficient matrix will be positive so that the matrix is non-singular. As a consequence the local problem has a unique solution and will reproduce $S_{p,\xi,I}$.

In the second step we set the coefficient c_j of the global approximation Pf' to b_j, so that

$$b_j = c_j.$$

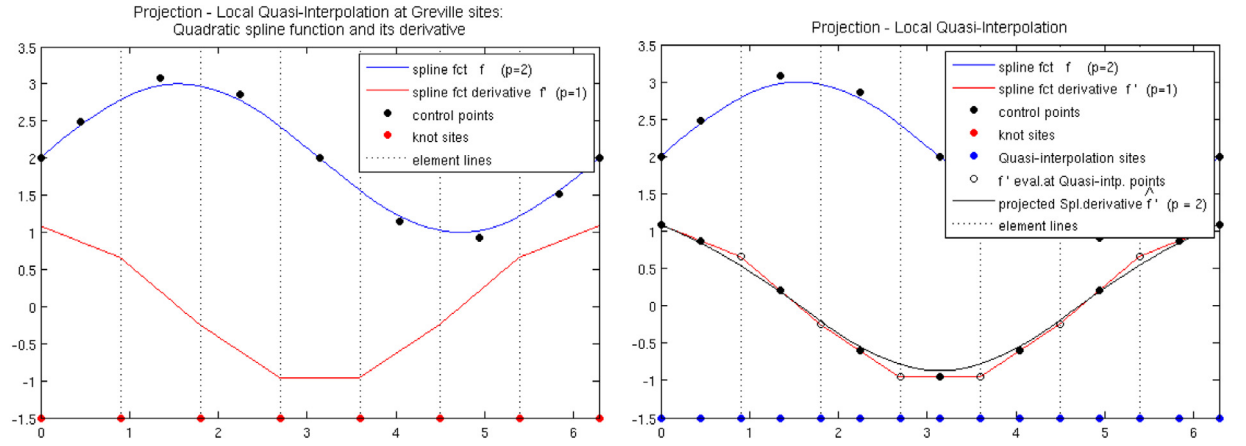

Fig. 9. Projection using local method quasi-interpolation: **Left:** Quadratic spline function $f \in \mathbb{S}_{2,\Xi}$ (blue) and its linear derivative $f' \in \mathbb{S}_{1,\Xi}$ (red). **Right:** Quasi-interpolation sites are computed illustrated as blue dots using Eq. (18); spline derivative $f' \in \mathbb{S}_{1,\Xi}$ evaluated at quasi-interpolation points (clear dots); control point set (black dots); projected spline derivative $\hat{f}' \in \mathbb{S}_{2,\Xi}$ (black) representing a quadratic spline. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

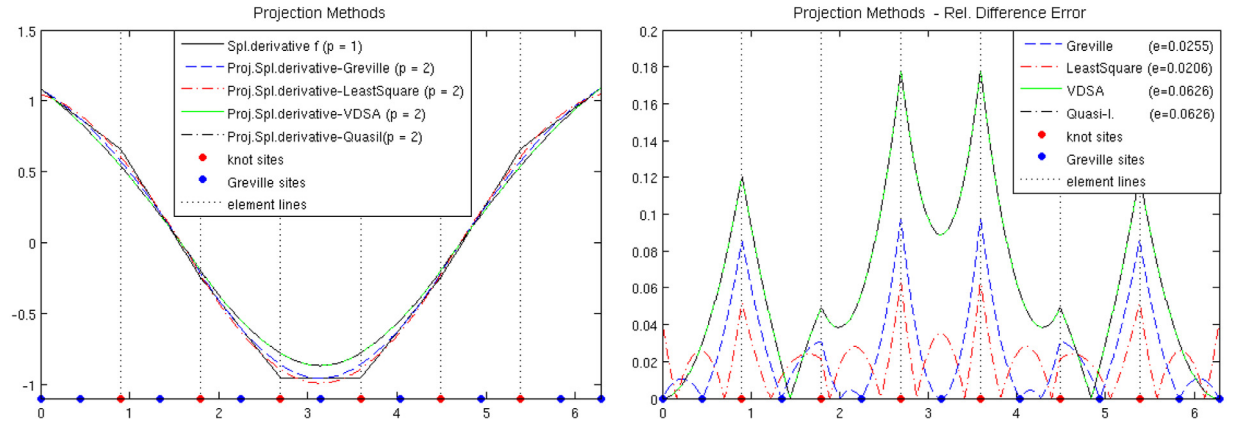

Fig. 10. Projection methods in comparison: **Left:** Spline derivative $f' \in \mathbb{S}_{1,\Xi}$ (black); projection exploiting Greville interpolation method (blue); projection exploiting least-squares method (red); projection exploiting VDSA (green); projection exploiting quasi-interpolation (dashed black). **Right:** Relative difference between spline derivative $f' \in \mathbb{S}_{1,\Xi}$ and projection $\hat{f}' \in S_{p=2}$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The spline $Pf' = \sum_{j=1}^{n} N_{j,p} b_j = \hat{f}' \in S_{p,\Xi}$ will then be the approximation to $f'(\xi) \in S_{p-1,\Xi}$.

We plotted in Fig. 9 an example case for degree $p = 2$. Note, evaluation of a linear spline function at quasi-interpolation points results in VDSA control values and the projection generates therefore the same spline derivative of degree 2 as the VDSA projection method.

2.4. Summary

In the previous sections we derived four different projection approaches in order to present the secondary results within the same spline space as the primary (displacement) results. We looked at global and local interpolation and approximation methods.

We discussed for each method the same example based on a smooth quadratic spline curve with linear derivative. We retrieved the quadratic derivative by projection. The linear spline derivative $f' \in S_{p=1}$ with its quadratic projections $\hat{f}' \in S_{p=2}$ using the presented local and global projection methods are shown in the left image of Fig. 10. The two local methods are identical due to the special case of $p = 2$ where the VDSA projection gives the same results as the three point quasi-interpolation method. The shape preserving properties (cf. (86)) of the VDSA projection are

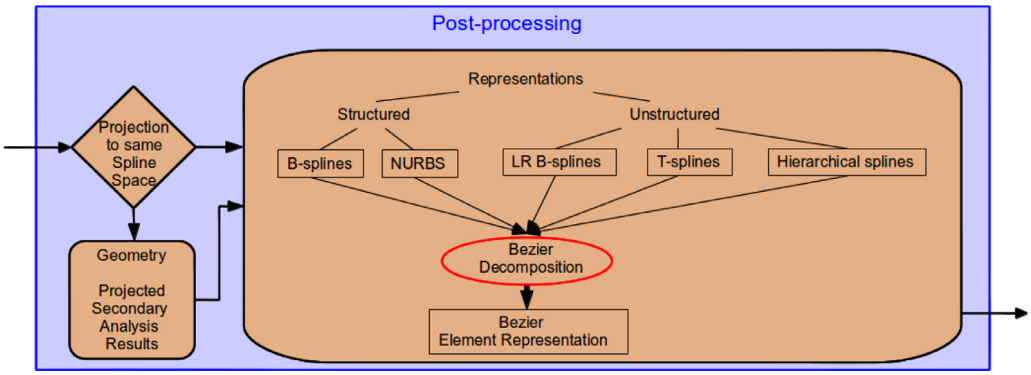

Fig. 11. Post-processing: Bézier Decomposition — Transform structured and unstructured representations into a Bézier element representation (generalized representation).

clearly maintained. In contrast, the least-squares approach generates new extreme values by optimizing the squared distance to the linear function in the quadratic case. The spline projection method exploiting the interpolation at the Greville points are the only interpolation method that matches the linear derivative exactly at the Greville points.

In the right image of Fig. 10 we plotted the relative difference measure as defined by (52) between $f' \in S_{p=1}$ and $\hat{f}' \in S_{p=2}$ to investigate the quality of reproduction of the derivative with the new basis function set one order higher. Both global methods are closest to the linear derivative function, where the least-squares method produces the best fit with a relative difference of 0.0206 closely followed by the global interpolation method with a relative difference of 0.0255. Both local methods measure a relative difference of 0.0626.

We summarize that both approximation/interpolation methods (results are shown at the end of Appendix A.2) and projection methods with respect to the relative difference measure and exploited method show no significant difference in performance. Whereas the global least-squares method performs superior to the spline interpolation method.

In order to evaluate and compare these methods even further under different aspects like performance of difference error measures, convergence, conditioning and cost we apply different examples and test cases in one, two and three-dimensions and present these results in Section 5.2. We mention properties, benefits and drawbacks and rise a discussion in Section 5.2.9. Finally, we give some concluding remarks in Section 6.

3. Post-processing: Bézier decomposition

In this section we explain how we obtain a unified Bézier element representation for structured and unstructured meshes in an efficient way. The use of Bézier elements facilitates parallel processing within the visualization task. The general steps and divisions are explained in detail below. This is also illustrated in Fig. 11 which is a close-up of the Post-processing part within Fig. 3.

The technique of Bézier decomposition [12,16] is a geometric modeling tool allowing to transform structured mesh representations, like B-Splines, NURBS and unstructured mesh representations, like LR B-splines and T-splines into a Bézier element representation. In the following section we describe how these functions can be exploited for efficient visualization.

3.1. Preliminaries

The mathematical model for the visualization is determined by the underlying geometric representation. A successful geometric design and shape representation in a parametric form is commonly formulated using parametric functions like, Bézier, B-spline or NURBS functions.

Before explaining their use for an efficient visualization pipeline we refer to the basic definitions for B-splines and LR B-splines given in Appendices A.1 and A.3, respectively. Note that these basic definitions are also used within Section 3.2 where we are concerned with the development of the Bézier decomposition for LR B-splines.

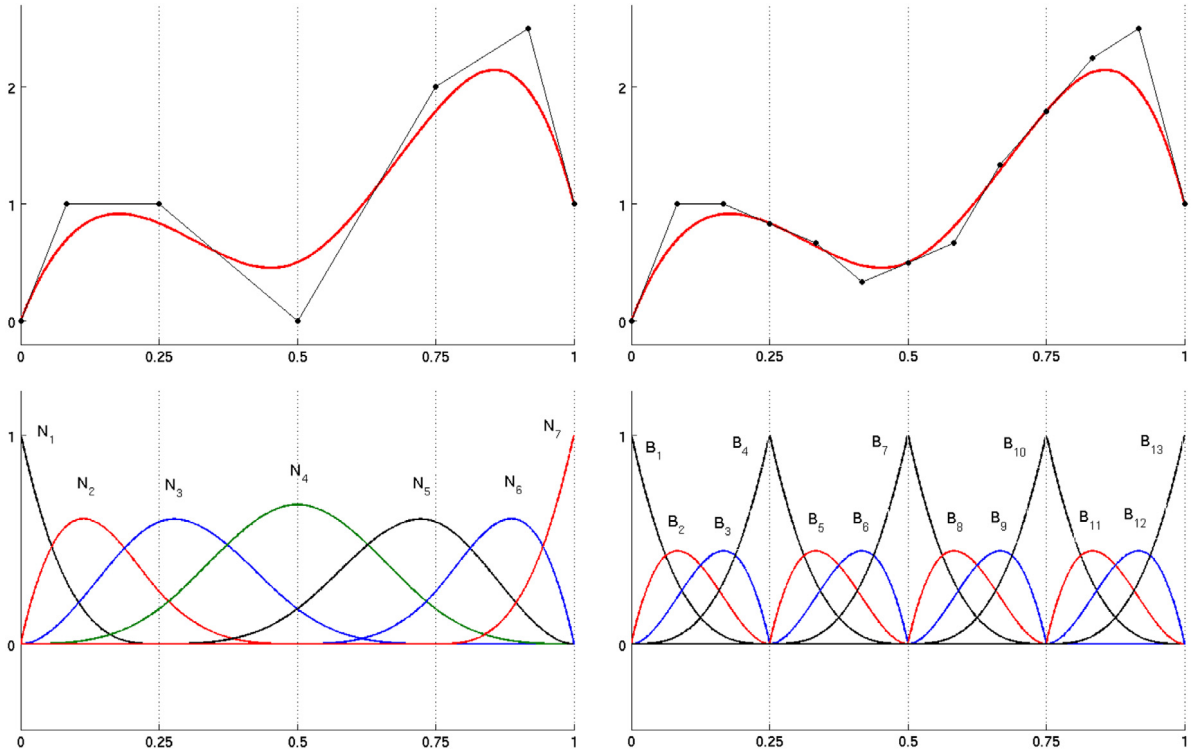

Fig. 12. **Left:** Cubic B-Spline with knot vector $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{11}\} = \{0, 0, 0, 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1, 1, 1, 1\}$ and 7 degrees of freedom. **Right:** After repeated knot insertion we obtain Bernstein basis functions and C^0-continuity at the Bézier element boundaries ($\tilde{\Xi} = \{0, 0, 0, 0, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{3}{4}, \frac{3}{4}, \frac{3}{4}, 1, 1, 1, 1\}$ and 13 degrees of freedom).

3.1.1. Bézier decomposition for structured meshes

In this section we consider the basic theory of Bézier decomposition for B-spline curves. Bézier decomposition is a special case of knot insertion, where curves, surfaces and volumes are decomposed into its constituent (Bézier) polynomials. These parts represent the so-called Bézier elements. Each interior knot of the original knot vector Ξ is repeatedly inserted until it has multiplicity p [12] allowing for the division of the curve into independent segments as illustrated in Fig. 12.

Bézier decomposition for NURBS was presented by Borden et al. [8]. We exploit this algorithm and apply it to univariate B-Splines in order to demonstrate the basic idea for structured representations. We assume that we are given a knot vector $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{p+n+1}\}$ and a set of control points P. In order to decompose the B-spline curve the m new knots $\{\hat{\xi}_1, \hat{\xi}_2, \ldots, \hat{\xi}_m\}$ are inserted. For each new knot $\hat{\xi}_j$ that is inserted we obtain a α_i^j, $i = 1, 2, \ldots, n + j$ as defined in (70). The Bézier extraction operator $C_j \in \mathbb{R}^{(n+j-1)\times(n+j)}$ can then be written as follows

$$C_j = \begin{pmatrix} \alpha_1 & 1-\alpha_2 & 0 & & \cdots & 0 \\ 0 & \alpha_2 & 1-\alpha_3 & 0 & \cdots & 0 \\ . & & \ddots & & & \\ 0 & \ldots & 0 & \alpha_{(n+j-1)} & 1-\alpha_{(n+j)} & \end{pmatrix}. \tag{21}$$

The new control points are computed recursively

$$\hat{P}_{j+1} = C_j^\top \hat{P}_j \tag{22}$$

with $\hat{P}_1 = P$ as initialization. Defining

$$C^\top = C_m^\top C_{m-1}^\top \cdots C_1^\top \tag{23}$$

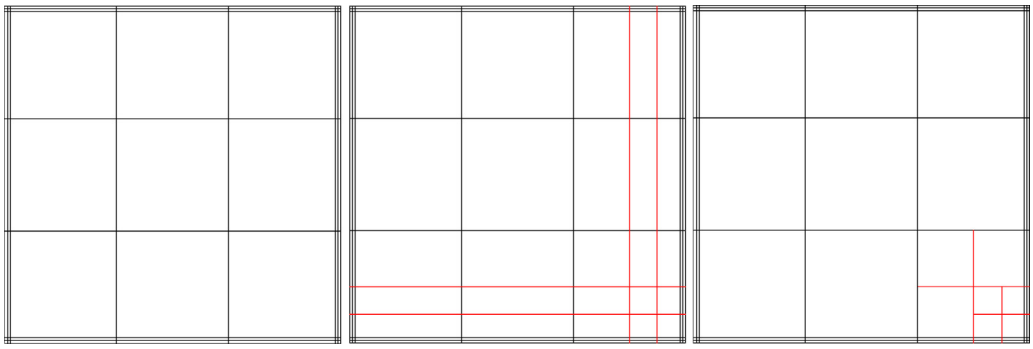

Fig. 13. Refinement techniques: It is desired to refine the lower right corner element (first image). Tensor product B-spline refinement using knot insertion leads to a global refinement structure (second image). "True" local refinement — only refining the region on demand using for example LR B-splines.

and $P^* = \hat{P}_{m+1}$ as the final set of control points we get

$$P^* = C^\top \hat{P}, \tag{24}$$

where P has dimension $n \times d$, C has dimension $n \times (n+m)$ and P^* has dimension $(n+m) \times d$.

Exploiting the fact that knot insertion does not change the parametric or geometric representation, a curve $F(\xi)$ consisting of Bernstein basis functions $B(\xi)$ can be formulated as follows

$$\begin{aligned} F(\xi) &= P^* B(\xi) && (25)\\ &= (C^\top P)^\top B(\xi) && (26)\\ &= P^\top C B(\xi) && (27)\\ &= P^\top N(\xi). && (28) \end{aligned}$$

With this formulation it is shown in [8] that a new basis and linear operator is constructed such that

$$N(\xi) = C B(\xi). \tag{29}$$

Note that the extraction operator C does not depend on the basis functions or control points but only on the knot vector. This allows the direct application of C to B-splines resulting in their Bézier element representation.

In Fig. 12 we have displayed a cubic B-spline curve with maximum continuity C^2 and the resulting Bézier spline curve along with the basis functions and new control points after knot insertions. Note that even though the Bernstein basis in general is C^0-continuous it may very well represent exactly a higher continuous spline function of the same polynomial order by "proper" choice of the control points. The Bézier extraction operator defined above ensures this "proper" choice of the control points.

3.1.2. Bézier decomposition for unstructured meshes

Local unstructured refinement techniques with respect to splines can be classified into three categories [17]. The first category is based on the concept of partition of unity. This includes for example local refinement techniques like LR B-splines [2] (cf. Fig. 13) and T-splines [5]. The second category also inheriting the partition of unity property comprises hierarchical refinement techniques, like the approaches from Vuong et al. [6] and Schillinger et al. [7]. The third category is based on subdivision refinement techniques pursuing a different data structure than the other two element based techniques. A basis function is decomposed into the sum of refined basis functions of the same class and is then replaced by the new basis functions. Therefore, only the methods of the first two categories are adequate mesh representations for Bézier decomposition operations.

The technique of Bézier decomposition for T-splines was introduced by Scott et al. [9]. However, for LR B-splines the Bézier decomposition is not well documented elsewhere. Thus, we present this technique in order to explain the transformation that helps us to generate a Bézier element representation also for LR B-splines — to generalize

the visualization pipeline. The basic technique of locally refined B-splines are presented in Appendix A.3 for completion.

3.2. Bézier decomposition for LR B-splines

Each LR B-spline basis function has an exact representation over each element in terms of Bézier basis functions, known as the previous mentioned Bernstein polynomials $B(\xi)$ stated in equation Eq. (55). As a consequence, each locally defined LR B-spline basis function $N_{\Xi_i}(\xi) = N_{p,\Xi_i}(\xi)$ is a linear combination of Bézier basis functions, thus for each $N^e_{\Xi_i}(\xi)$ over element e there exist coefficients $c^e_{\Xi_i,k}$ so that

$$N^e_{\Xi_i}(\xi) = \sum_{k=1}^{p+1} c^e_{\Xi_i,k} B_k(\xi). \tag{30}$$

We write Eq. (30) in matrix–vector form

$$N^e_{\Xi_i}(\xi) = C^e_{\Xi_i}\ B(\xi), \tag{31}$$

where $C^e_{\Xi_i}$ is the respective extraction operator for the basis function $N^e_{\Xi_i}(\xi)$ for each element e.

3.2.1. The element extraction operator

In order to decompose the LR B-spline basis functions over each element e into Bernstein polynomials the multiplicity of the knots referring to that element have to be verified. Consecutive knot insertion using Eqs. (94) and (95) for each individual LR B-spline basis function which has support on that particular element is performed until C^{-1} continuity at the element knots is reached, so that the multiplicity of the knots at the element boundaries is $p+1$ and $q+1$ respectively.

For each inserted knot ξ_j a univariate extraction operator $C^e_{\Xi_i,\xi_j}$ is built with respect to the basis function which has support in element e with the following matrix

$$C^e_{\Xi_i,\xi_j} = \begin{pmatrix} \alpha_1 & \alpha_2 & 0 & \cdots & & & 0 \\ 0 & \alpha_3 & \alpha_4 & 0 & \cdots & & 0 \\ 0 & 0 & \alpha_5 & \alpha_6 & 0 & \cdots & 0 \\ \vdots & & & \ddots & & & \vdots \\ 0 & & \cdots & & 0 & \alpha_{2m-1} & \alpha_{2m} \end{pmatrix}. \tag{32}$$

We obtain the vector-valued univariate element extraction operator for element e in the first parameter direction for one particular basis function by

$$C^e_{\Xi_i} = C^e_{\Xi_i,\xi_1} C^e_{\Xi_i,\xi_2} \cdots C^e_{\Xi_i,\xi_m}, \tag{33}$$

analogous done also for the other parameter directions.

Similarly to NURBS, multi-variate extraction operators of LR B-splines can be computed as products of univariate extraction operators $C^e_{\Xi_i}$ and $C^e_{\mathcal{H}_i}$ so that

$$N^e_{\Xi_i,\mathcal{H}_i} = [C^e_{\Xi_i} B^e_{\Xi_i}(\xi)][C^e_{\mathcal{H}_i} B^e_{\mathcal{H}_i}(\eta)]. \tag{34}$$

A major difference between NURBS and LR B-splines is that where NURBS have a global parametrization, LR B-splines have a local parametrization. This leads to two differences in how the extraction operators are computed. First, with LR B-splines, we work with local knot vectors that are in general not open, that means the first and last knot may have multiplicity less than $p+1$.

Local knot vectors correspond to individual functions. As a local knot vector is processed we compute a single row for each of the corresponding element extraction operators as opposed to compute the entire element extraction operator at once.

We obtain for each particular basis function one vector-valued extraction operator building one row in the full extraction operator matrix C^e for element e. That means, the entire process is repeated for each basis function which

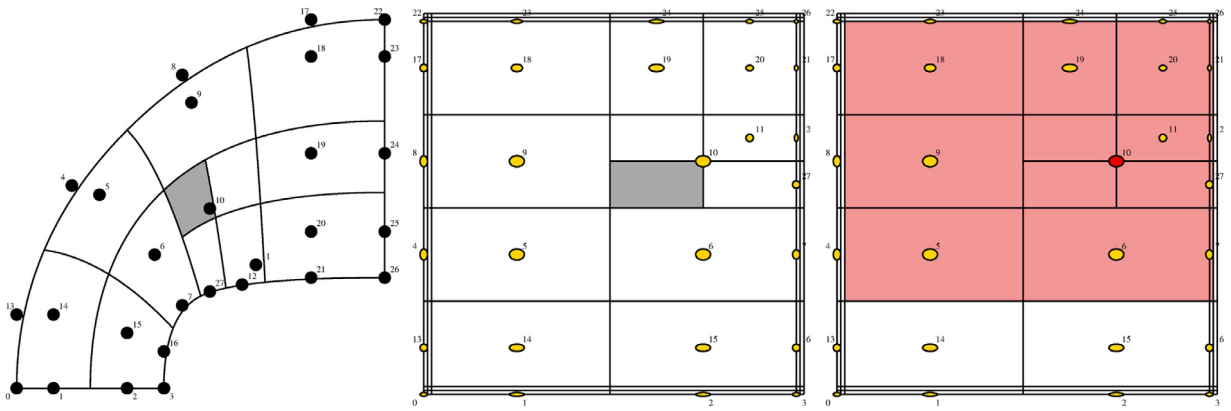

Fig. 14. Refined example: We extract the Bézier basis functions for a selected LR B-spline basis function which has support over a particular element. **Left:** Highlighted element e_6 in the refined LR-mesh with respective control points. **Center:** Highlighted element e_6 in the parameter domain with respective basis functions. **Right:** The red shading and red dot indicate the LR B-spline basis function $N_{\Xi_{10},\mathcal{H}_{10}} = [0122][\frac{1}{2}1\frac{3}{2}2]$ which has support over the highlighted element e_6. We generate for this particular LR B-spline basis function the respective vector-valued extraction operator. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

has support over element e resulting in an element extraction operator matrix

$$C^e = \begin{bmatrix} \tilde{\alpha}_{i,1}\tilde{\beta}_{i,1} & \tilde{\alpha}_{i,1}\tilde{\beta}_{i,2} & \cdots & \tilde{\alpha}_{i,1}\tilde{\beta}_{i,q+1} & \tilde{\alpha}_{i,2}\tilde{\beta}_{i,1} & \cdots & \tilde{\alpha}_{i,p+1}\tilde{\beta}_{i,q+1} \\ \tilde{\alpha}_{j,1}\tilde{\beta}_{j,1} & \tilde{\alpha}_{j,1}\tilde{\beta}_{j,2} & \cdots & \tilde{\alpha}_{j,1}\tilde{\beta}_{j,q+1} & \tilde{\alpha}_{j,2}\tilde{\beta}_{j,1} & \cdots & \tilde{\alpha}_{j,p+1}\tilde{\beta}_{j,q+1} \\ \vdots & & & & & & \vdots \\ \tilde{\alpha}_{k,1}\tilde{\beta}_{k,1} & \tilde{\alpha}_{k,1}\tilde{\beta}_{k,2} & \cdots & \tilde{\alpha}_{k,1}\tilde{\beta}_{k,q+1} & \tilde{\alpha}_{k,2}\tilde{\beta}_{k,1} & \cdots & \tilde{\alpha}_{k,p+1}\tilde{\beta}_{k,q+1} \end{bmatrix}, \tag{35}$$

where the subscript i indicates the particular LR B-spline basis function $N_{\Xi_i,\mathcal{H}_i}$. The size of this matrix is of dimension $N_e \times P_e$, where N_e is the number of LR B-spline basis functions having support on element e and where $P_e = (p+1)(q+1)$.

Note, during the knot insertion new basis functions can be created which have no support on the considered element. Only corresponding α-values, which belong to a basis function which has support over the particular element are entries of the element extraction operator matrix.

We demonstrate this process exploiting a refined version (cf. Fig. 14) of the example shown in Fig. 69. We select one element $e_6 = ([\xi_a, \xi_b]; [\eta_a, \eta_b]) = ([1, \frac{3}{2}]; [1, \frac{3}{2}])$ (highlighted in gray in the LR-mesh), where ξ_a and ξ_b are the element boundaries in the first parameter direction and η_a and η_b in the second. We generate for LR B-spline basis function $N_{\Xi_{10},\mathcal{H}_{10}} = [0122][\frac{1}{2}1\frac{3}{2}2]$ which has support over that particular element (indicated in red in the right image of Fig. 14) the element extraction operator to obtain the respective set of Bernstein polynomials.

We start by verifying which knots have to be inserted in the first parameter direction of the LR B-spline basis function $N_{\Xi_{10},\mathcal{H}_{10}} = [0122][\frac{1}{2}1\frac{3}{2}2]$ to achieve C^{-1}-continuity at the element boundaries $[\xi_a, \xi_b] = [1, \frac{3}{2}]$ for element e_6. Note, in the following we consider only the first parameter direction of the LR B-spline basis function. The process for the second parameter direction is done analogous.

The following knots $(\xi_1, \xi_2, \xi_3, \xi_4, \xi_5) = (1, 1, \frac{3}{2}, \frac{3}{2}, \frac{3}{2})$ have to be inserted one after the other into the LR B-spline basis function $N_{\Xi_{10}} = [0122]$. The insertion-tree which develops by doing so is given by Fig. 15 where we perform repeatedly knot insertion using Eqs. (93)–(95) and generate the respective extraction operator by placing the computed α-values into the operator matrix (32).

Using

$$\tilde{C}^e_{\Xi_{10}} = \tilde{C}^e_{\Xi_{10},\xi_1}\tilde{C}^e_{\Xi_{10},\xi_2}\tilde{C}^e_{\Xi_{10},\xi_3}\tilde{C}^e_{\Xi_{10},\xi_4}\tilde{C}^e_{\Xi_{10},\xi_5} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{3}{4} & \frac{5}{8} & \frac{5}{8} & \frac{1}{2} \end{bmatrix} \tag{36}$$

we obtain the element extraction operator for the first parameter direction for basis function $N_{\Xi_{10}}$.

During the knot insertion new basis functions are created which have no support on the considered element e_6. We indicate in Fig. 15 with red, basis functions which have support on e_6. Only corresponding α values, which belong to

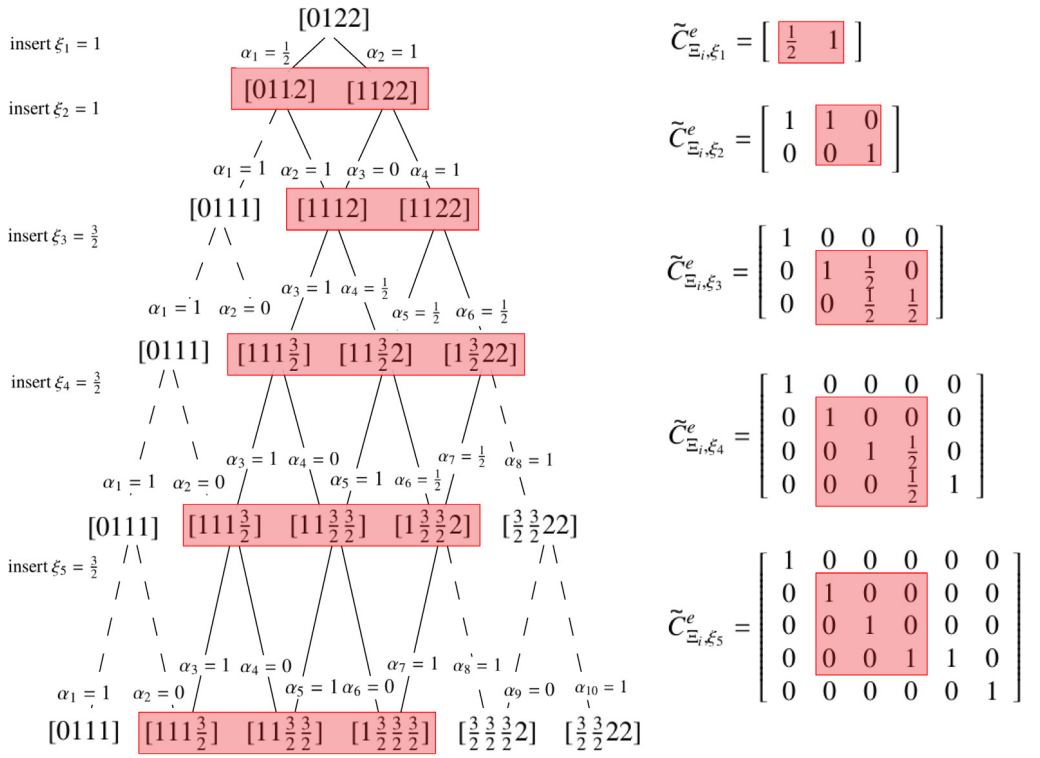

Fig. 15. Knot insertion — Tree: The local knot vector (first parameter direction) of the LR B-spline basis function is the root of the tree. One knot is inserted into the local knot vector. Two α-values are computed using Eqs. (94) and (95). These α-values are placed into the respective extraction operator for that knot insertion operation. After the insertion two new basis functions emerge. The process of knot insertion for these local knot vectors is repeated until C^{-1}-continuity at the element boundaries $[\xi_a, \xi_b] = [1, \frac{3}{2}]$ is reached.

these basis functions are entries of the element extraction operator C^e. That means we obtain an in dimension-reduced vector-valued extraction operator $C^e_{\Xi_{10}} = [\hat{\alpha}_1\hat{\alpha}_2\hat{\alpha}_3]$ so that Eq. (30)

$$N^e_{\Xi_{10}}(\xi) = C^e_{\Xi_{10}} B(\xi) = \begin{bmatrix} \frac{1}{2} \frac{3}{4} \frac{5}{8} \end{bmatrix} \begin{bmatrix} B_2 \\ B_3 \\ B_4 \end{bmatrix} \tag{37}$$

for element e^6 for $N^e_{\Xi_{10}}(\xi)$ is satisfied — instead of the pre-computed operator matrix $\tilde{C}^e_{\Xi_{10}}$ which contains all α values even those which have no support on e_6 (cf. Fig. 15 and Fig. 14 right hand side). These particular operators are illustrated in Fig. 15 and the resulting Bernstein basis functions are illustrated in the left image of Fig. 16.

Repeating the procedure for the second parameter direction we obtain $C^e_{\mathcal{H}_{10}} = [\hat{\beta}_1\hat{\beta}_2\hat{\beta}_3]$ and for basis function $N^e_{\Xi_{10}\mathcal{H}_{10}}(\xi, \eta)$ we obtain one row in the full extraction operator C^e (35) for element e.

3.2.2. *Partition of unity — Scaling by weights*

One of the known B-spline properties can be formulated as follows: A spline lies in the convex hull of its control points if the B-splines form a partition of unity. After performing the local refinement the partition of unity will in general be violated. That means we have to make adjustments to the new computed basis functions which are created after the knot insertion process. In order to keep the partition of unity property for the new set of basis functions we choose the process of scaling by weights as suggested by Dokken et al. [2]. For all levels of refinement the partition of unity

$$\sum_i \gamma_i N_{\Xi_i}(\xi) = 1 \tag{38}$$

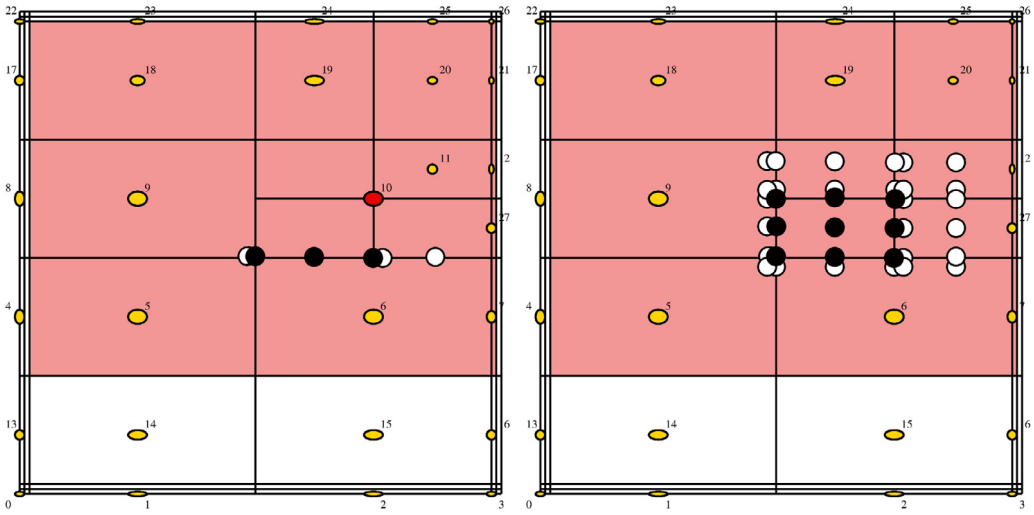

Fig. 16. **Left:** Bernstein functions indicated in the first parameter direction as dots. Black dots indicate the basis functions which have support on element e^6. **Right:** All Bernstein functions which are extracted on element e^6.

with respect to all basis functions in each particular refinement level has to be satisfied. This implies for the LR B-spline function f that

$$f = \sum_i \gamma_i N_{\Xi_i}(\xi) P_i. \tag{39}$$

3.2.3. New set of control points

In order to compute the new set of basis functions and control points over each particular element e we have to consider the respective weight γ_i for each individual LR B-spline basis function $N_{\Xi_i,\mathcal{H}_i}$ having support on element e to keep the partition of unity. Thus we define the weighted Bézier element extraction operator C_γ^e as

$$C_\gamma^e = \begin{bmatrix} \gamma_i\tilde{\alpha}_{i,1}\tilde{\beta}_{i,1} & \gamma_i\tilde{\alpha}_{i,1}\tilde{\beta}_{i,2} & \cdots & \gamma_i\tilde{\alpha}_{i,1}\tilde{\beta}_{i,q+1} & \gamma_i\tilde{\alpha}_{i,2}\tilde{\beta}_{i,1} & \cdots & \gamma_i\tilde{\alpha}_{i,p+1}\tilde{\beta}_{i,q+1} \\ \gamma_j\tilde{\alpha}_{j,1}\tilde{\beta}_{j,1} & \gamma_j\tilde{\alpha}_{j,1}\tilde{\beta}_{j,2} & \cdots & \gamma_j\tilde{\alpha}_{j,1}\tilde{\beta}_{j,q+1} & \gamma_j\tilde{\alpha}_{j,2}\tilde{\beta}_{j,1} & \cdots & \gamma_j\tilde{\alpha}_{j,p+1}\tilde{\beta}_{j,q+1} \\ \vdots & & & & & & \vdots \\ \gamma_k\tilde{\alpha}_{k,1}\tilde{\beta}_{k,1} & \gamma_k\tilde{\alpha}_{k,1}\tilde{\beta}_{k,2} & \cdots & \gamma_k\tilde{\alpha}_{k,1}\tilde{\beta}_{k,q+1} & \gamma_k\tilde{\alpha}_{k,2}\tilde{\beta}_{k,1} & \cdots & \gamma_k\tilde{\alpha}_{k,p+1}\tilde{\beta}_{k,q+1} \end{bmatrix} \tag{40}$$

and compute the new set of basis functions $\hat{P}^e$ as follows

$$\hat{P}^e = (C_\gamma^e)^\top P^e, \tag{41}$$

where P^e is a set of control points of LR B-spline basis functions having support on element e.

We proceed for each element and obtain the Bézier basis function representation of the LR B-spline surface as shown in Fig. 17.

With having all the post-processing stages, projection and Bézier decomposition in place we will focus in the following section on the visualization part of the new IGA visualization pipeline.

4. Scientific visualization

During rendering a two-dimensional view of a three-dimensional scalar field is generated. The scalar field represents typically the result of an isogeometric simulation like the von Mises stress. Conventional methods exploit tessellation algorithms in order to convert the extracted boundary surfaces of a volume into a set of triangles. Spline surfaces are directly evaluated at predefined or adapted sampling points (which define the resolution of the 2D view)

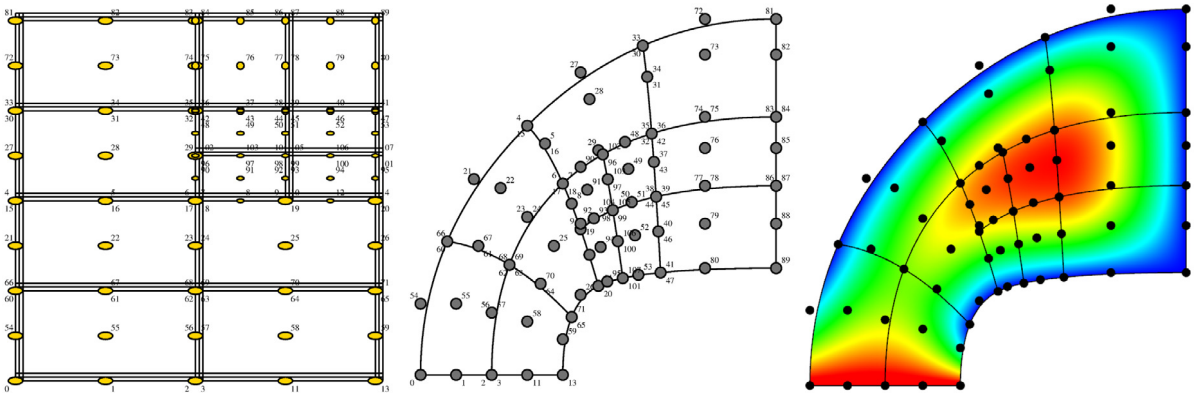

Fig. 17. Bézier representation of the LR B-spline surface. Note that the double corner control points of element e_6 are interpolatory at the edge of element e^6 but they are not interpolatory at the center point of element e^5.

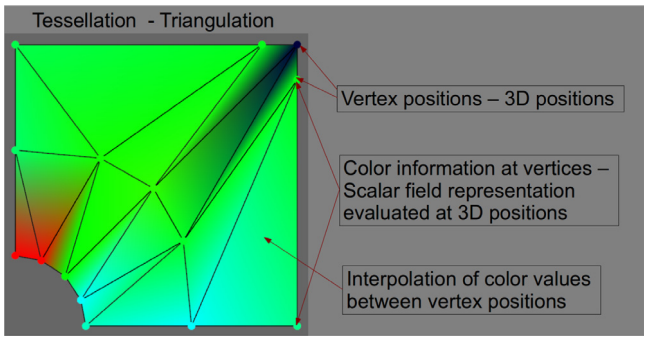

Fig. 18. Tessellation — Triangulation: Surfaces of a volume are extracted into a set of triangles by choosing a certain sampling rate of evaluation points. The spline surface is evaluated at the triangle edges — 3D positions, so called vertex positions. The scalar field (here von Mises stress) values determine color at the triangle vertices. Pixel color values between vertices are then linearly interpolated over each triangle.

so called vertex positions. Corresponding color values, with respect to analysis results are then linearly interpolated over each triangle. Using this type of rendering methods the correct color is only defined at the vertices (cf. Fig. 18).

In order to prevent the appearance of shading artifacts that are caused by interpolation during the static visualization pipeline (cf. left image of Fig. 19 where the visualization interpolation procedure is performed without directly programming on graphics hardware [shader programming]), one might use a per-pixel evaluation of the interpolation values exploiting vertex and fragment programming. Vertex positions still have the same color value, but a per-pixel evaluation of the interpolation between the vertex positions is performed within the fragment shader program as shown in the center image of Fig. 19.

In order to perform a pixel-accurate representation one faces the challenge to subdivide the tessellation area using error correction terms, where each triangle only covers a few pixels [18,19]. For high quality visualization a direct rendering approach can be exploited that evaluates the color for each pixel. This can be achieved by using ray based intersection methods like the here used ray casting rendering technique. The result for the von Mises stress representation is depicted on the right hand side of Fig. 19.

A real-time computation of the visualization can be performed exploiting GPUs. In the following section we are concerned with current rendering techniques and explain the method we implemented for the visualization.

4.1. Rendering techniques

Current visualization techniques that are used within applied sciences can be divided into three different branches: Visualization methods based on *direct volume rendering*, *direct rendering of isosurfaces* and *isosurface extraction algorithms*.

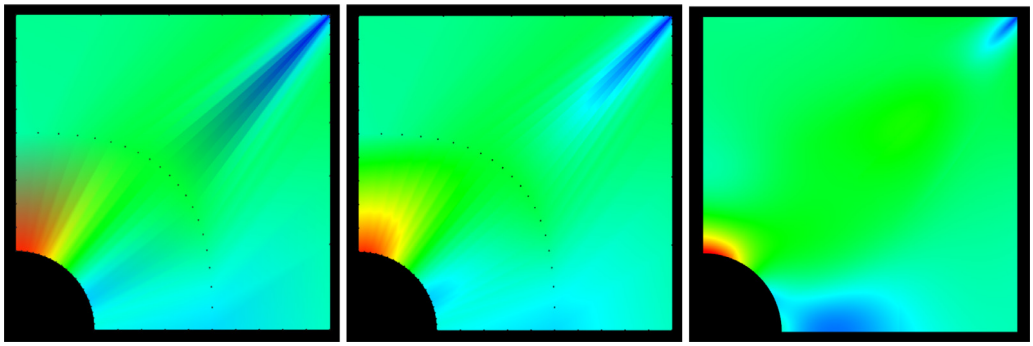

Fig. 19. Scalar field visualization without and with shader programming: **Left:** Conventional, scalar values of a solution field (here von Mises stress) are evaluated at the black marked vertex positions. Utilizing these vertex positions a tessellation of the surface is generated. Linear interpolation within each tessellation triangle of the scalar values at the vertex positions leads to shading effects (gray overlay). **Center:** Vertex shader programming. Tessellation vertices are passed to the vertex shader program and a per-pixel interpolation of the scalar field values given in the vertices was exploited. We obtain an improvement to the non-shader method: Clearer colors, no gray overlays. **Right:** Fragment shader programming. A per-pixel evaluation of the scalar field takes place using ray based intersection methods, like ray casting or ray tracing. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Isosurface extraction methods like Marching Cube (MC) techniques [20] were first introduced as an efficient method for 3D visualization of medical data sets which were obtained by Magnetic Resonance Imaging (MRI) or Computer Tomography (CT). They are performed on a regular grid structure and extract isosurfaces exploiting a list of candidate polygons resulting in a piecewise planar approximation of the isosurface. An improvement of the MC algorithm was published using a splitting box algorithm [21] with the idea to create a smaller number of candidate polygons than the MC by recursively subdividing the volume until a certain complexity property is reached. For structured as well as for unstructured grids Marching Tetrahedra (MT) [22] can be used. In the MC and MT methods element vertices are utilized to determine if an isosurface passes through the respective element. Element edges and isosurfaces are exploited to create a facet structure approximating the isosurface. These methods are efficient and widely used, but due to the nature of their approximation, the topological correctness cannot be guaranteed.

Therefore, more advanced techniques like direct isosurface rendering belonging to the branch of direct rendering methods were introduced (cf. [23]). It allows to directly visualize an isosurface from trivariate NURBS of arbitrary geometry. For each pixel in the image plane a corresponding point on the isosurface is computed by the intersection between a ray frustum and the isosurface. The pixel can be shaded using the gradient as normal for the given point [24,23].

Direct volume rendering techniques like ray tracing methods [25] are also exploited in the gaming industry where it becomes more and more important to generate a very high degree of visual realism. Therefore, specular reflectance, reflectance or attenuation and absorption depending on the material properties of the objects are computed. Rays are constructed along the line of sight starting from the center of the camera. It is necessary to integrate each ray through the volume. Expensive root solving techniques along with the previous mentioned surface appearances of the object makes this method computationally extremely costly.

Requiring a very high rendering quality of complex primitives like parametric surfaces and volumes we decided to exploit a volume rendering technique based on the family of ray casting methods [26–28]. There for each pixel a ray – starting at the camera center and passing through pixel in the image plane – is generated. The RGB color values are evaluated for each display-pixel in the full image plane and the ray may be clipped by volume boundaries for efficiency purposes. Volume ray casting is considered to result in the highest visualization quality. Below we explain this method in more detail.

4.2. GPU based shader programming

Over recent years graphics processors have changed drastically, while at first one was dependent on a fixed functionality in the graphics pipeline as depicted in Fig. 20. For programming using the OpenGL specifications that means that one would have just a single lighting model available. Thus, lighting calculations would only be done at the vertex positions and then interpolated over the primitive by the fixed function fragment processor. Nowadays, the

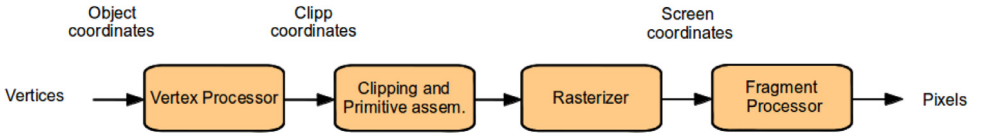

Fig. 20. Fixed-function graphics pipeline.

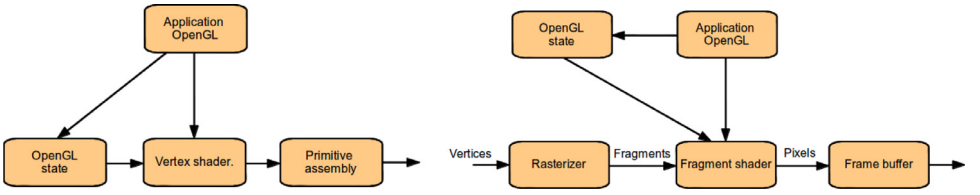

Fig. 21. User programmable vertex and fragment processor writing vertex and fragment shader respectively: **Left:** Vertex shader pipeline. **Right:** Fragment shader pipeline.

vertex as well as the fragment processors are user programmable. A program which is written for graphics hardware is called a shader. Mainly, there are two basic types of shader programs, the vertex shader – intended to run on the vertex processor – and the fragment shader – intended to run on the fragment processor. In order to achieve a high accuracy in the visualization result vertex and fragment shaders can be written individually. Vertex as well as fragment shaders operate in a parallel architecture and can therefore apply transformations simultaneous to a large set of elements. Most modern GPUs have multiple shader pipelines to facilitate this, vastly improving the computation of the data flow.

Common tessellation algorithms are computed on the CPU. That implies that the tessellation information is kept in graphics memory and has to be updated continuously leading to high computational costs, furthermore, it increases the bandwidth for the data transfer to the GPU and as a consequence slows down the visualization performance. The strategy to keep tessellation information in GPU memory is therefore preferable, especially for large scale data sets.

In order to exploit the parallel computing power of GPUs we propose to use shader programming technology.

4.2.1. Shader programming language

Our approach for an efficient parallel implementation of the visualization is based on OpenGL [29] and the OpenGL Shading Language (GLSL) [30] — a high-level shading language that allows a more direct control of the graphics pipeline without having to use assembly language or hardware-specific languages.

In the following we describe the vertex and fragment shaders which we exploit within the visualization pipeline in order to improve its flexibility and ease its maintenance.

4.2.2. Vertex shader

A vertex program operates as an improved replacement for the fixed function operations performed by the early vertex processor as illustrated in Fig. 21. The operations are defined in the vertex shader which is executed for each vertex. The input of a vertex shader consist most often in the vertex position defined by the application program. For each input argument an output argument for the rasterizer has to be determined (for example the vertex position). The vertex positions are usually provided in object space (x, y, z world coordinates) and the operation in the vertex shader consists of the transformation of this vertex position into the eye, or camera coordinates by exploiting the model-view projection matrix. An additional operation projects this camera coordinates into the screen or clip coordinates, resulting for the complete operation into a matrix–vector multiplication as follows:

```
-------------------------------------------------------
/* vertex shader */
void main(void)
{
   gl_Position  = gl_ModelViewProjectionMatrix * gl_Vertex;
   gl_FronColor = gl_Color;
}
-------------------------------------------------------
```

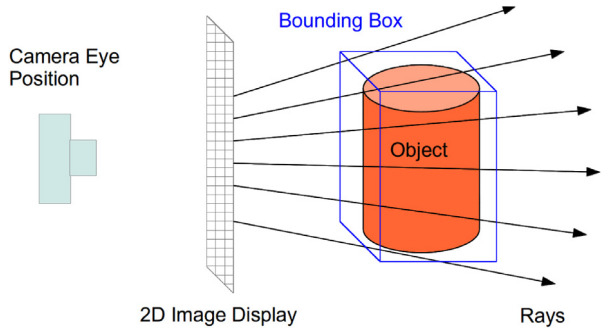

Fig. 22. Ray casting: Rays are constructed using (42). The ray volume patch intersection are then computed using equation (43).

The screen coordinate positions are stored in `gl_Position`. This version of a vertex shader program is one of the simplest ones where 3D object coordinates are transformed into screen coordinates and the color values at the 3D object coordinates from the application program are passed on to the vertex processor.

4.2.3. *Fragment shader*

Fragment shader programs are executed after the rasterizer as shown in Fig. 21 and that is the reason why the fragment shader operates on each fragment to compute the color value for each pixel of the screen. Vertex attributes, such as position or color are interpolated by the rasterizer across a primitive to generate the corresponding fragment attributes. An example for a basic fragment shader program is given here:

```
-------------------------------------------------------
/* fragment shader */
void main(void)
{
   gl_FragColor = gl_Color;
}
-------------------------------------------------------
```

The difference to the vertex shader program is that here the values of `gl_Color` are not the values produced by the vertex program. The values of `gl_Color` in the fragment shader program have been created by the rasterizer interpolating the values of `gl_FrontColor` from the vertex processor over the primitive to produce the values used in the fragment program. The fragment color produced by the fragment processor is used to modify the color of a pixel in the frame buffer as illustrated in Fig. 21.

4.2.4. *Ray-Casting — Pixel-accurate rendering*

In this section we explain our pixel-accurate rendering technique which is mainly based on fragment shader programming. We adopted the idea of the method from [27] for NURBS surfaces and extend it to unstructured mesh representations and volume rendering. As we can define any quantity at a vertex and interpolate it across a surface or volume element in a fragment shader we are also able to do an evaluation of geometry and simulation results on a fragment-by-fragment basis rather than on a vertex-by-vertex basis. Using this approach, for each pixel in the screen-space a ray is generated and ray object intersections are computed. At the intersection points the spline surfaces are evaluated. The user may define if a surface rendering or a full volume rendering of the object is desired with respect to his visualization requirements of the object.

Geometry as well as primary and (when required) projected secondary IGA results are stored in form of textures. Shader programs are generated depending on the polynomial degree of the geometry data. Primitives, like control points and respective scalar field values are passed to the vertex shader – in parallel – within the vertex shader the convex hull (bounding box cf. Fig. 22) for each Bézier patch of its control points is computed (cf. Algorithm 1). This is used within the later executed ray-casting algorithm.

The vertex shader program provides the homogeneous positions of the coordinates in screen space to the fragment shader program. Varying variables are passed on from the vertex processor to the fragment processor. The per-pixel

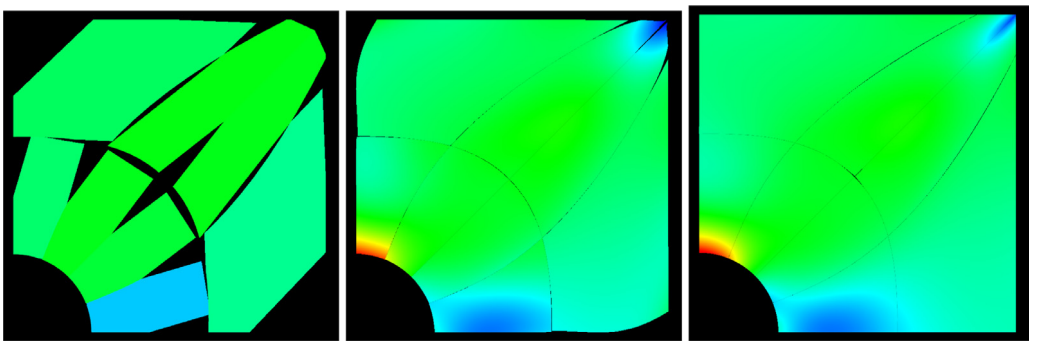

Fig. 23. Newton iteration: Results after one, two and three iterations. The converged result after 4 iterations is shown in the right image of Fig. 19.

Bézier value evaluation – exploiting a classical ray-casting approach – is performed in the fragment shader taking the convex hull area into account. At this stage also the color and the shading of the object is computed which leads to the final graphical output on the display. In the following the ray-casting procedure are described in more detail.

In order to generate the correct ray equation for each pixel/fragment the ray is represented by its implicit form [27] through the intersection of two planes for finding the surface intersection with a third plane. The three planes are represented as

$$P_1 = (n_1, d_1) \quad P_2 = (n_2, d_2) \quad P_3 = (n_3, d_3), \tag{42}$$

where n_1, n_2 and n_3 are the normal vectors and d_1, d_2 and d_3 are the distances of the planes to the origin. The ray surface-patch or ray volume patch intersection equation is then given by

$$F(u, v, w) = \begin{pmatrix} n_1 \cdot V(u, v, w) - d_1 \\ n_2 \cdot V(u, v, w) - d_2 \\ n_3 \cdot V(u, v, w) - d_3 \end{pmatrix} = 0. \tag{43}$$

Here $V(u, v, w)$ is the Bézier-volume element (patch). Exploiting Newton iterations we obtain the roots of $F(u, v, w)$ as the parameter values of the ray patch intersection by starting with an initial guess $(u, v, w)_0^\top$ for the Newton steps

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix}_{k+1} = \begin{pmatrix} u \\ v \\ w \end{pmatrix}_k - J^{-1} \cdot F(u_k, v_k, w_k). \tag{44}$$

The Jacobian J of F at $(u, v, w)^\top$ is given by

$$J = \begin{pmatrix} n_1 \cdot V_u & n_1 \cdot V_v & n_1 \cdot V_w \\ n_2 \cdot V_u & n_2 \cdot V_v & n_2 \cdot V_w \\ n_3 \cdot V_u & n_3 \cdot V_v & n_3 \cdot V_w \end{pmatrix}, \tag{45}$$

where V_u, V_v and V_w are the partial derivatives of V at $(u, v, w)^\top$. We compute the inverse of the Jacobian directly and the iteration terminates if a certain accuracy ϵ is reached. Fig. 23 shows the first three iterations for the volumetric hole problem. The Newton method converges and terminates after 4 iterations for this particular problem instance and the result is shown in the right image of Fig. 19. The Newton method converges quadratically if the initial guess is close to the solution. We observed for the presented examples that the number of Newton iterations varied between 4 to 16 iterations for an accuracy of 10^{-8}. Knowing the beam intersection parameters $(u, v, w)^\top$ for each fragment/pixel the deCasteljau algorithm [12] is used to determine the analytic scalar field values that we wish to visualize.

Vertex shader: In this section we briefly describe the main tasks that are performed within the vertex shader program that is used within the visualization pipeline. The vertex shader determines rays that potentially pass through a Bézier-patch by considering the bounding box that covers the patch. For this the bounding box of the control points is

Algorithm 1 Ray Casting

```
Determine bounding box of Bézier element
for Each pixel in the bounding box do
   Calculate ray (from viewpoint to the current pixel)
   Set ray-parameter that defines initial point on the ray
    Define ray-step-width
   while ray-parameter < endpoint and ray in bounding box do
      Get sample from volume
      Evaluate at parameter value and increment
      Advance position along the ray direction by ray-step-width to the next sample
   end while
end for
```

computed and the projection of its corner points is used to determine the 2D bounding box where the fragment shader computes the ray intersection with the Bézier-patch for each pixel.

Fragment shader: The ray-Bézier-patch intersection is implemented as part of the fragment program. This means that for example the Newton iterations (cf. Eq. (44)) for finding the roots of the intersection equation (43) are implemented there. In the following we provide a simplified (two dimensions) and basic code piece illustrating the fragment shader program performing the above described Newton iteration method. Computing the derivatives of the Bézier-patch $V(u, v)$ which appear as Vu and Vv within the shown fragment shader code lead to (not shown) lengthy expressions. In our implementation they are the result of a direct evaluation of the de Casteljau algorithm and depend on the desired degree of the Bézier-patch. The current considered ray is determined by the normals $n1$ and $n2$.

```
--------------------------------------------------------------------------------
/* Part of fragment shader (simplified) */
...
// Jacobian
float J11 = n1.x*Vu.x+n1.y*Vu.y+n1.z*Vu.z;
float J12 = n1.x*Vv.x+n1.y*Vv.y+n1.z*Vv.z;
float J21 = n2.x*Vu.x+n2.y*Vu.y+n2.z*Vu.z;
float J22 = n2.x*Vv.x+n2.y*Vv.y+n2.z*Vv.z;

float deter;
if (abs(J11*J22-J12*J21)<=0.00000001)
    deter = 0.0;
else
    deter = 1/(J11*J22-J12*J21);

// Inverse Jacobian
float invJ11=deter*J22;
float invJ12=-deter*J12;
float invJ21=-deter*J21;
float invJ22=deter*J11;

// Distance from origin to planes
float d1 = camPos.x*n1.x + camPos.y*n1.y + camPos.z*n1.z;
float d2 = camPos.x*n2.x + camPos.y*n2.y + camPos.z*n2.z;

// Intersection equation F(u,v) = 0
float Fu = n1.x*V.x+n1.y*V.y +n1.z*V.z - d1;
float Fv = n2.x*V.x+n2.y*V.y +n2.z*V.z - d2;
```

```
// Newton Iteration
evalvertex.x =  evalvertex.x - 1.0*(invJ11*Fu + invJ12*Fv);
evalvertex.y =  evalvertex.y - 1.0*(invJ21*Fu + invJ22*Fv);
...
--------------------------------------------------------------------------------
```

5. Numerical examples

With the numerical examples of this section we intend to examine and illustrate the key properties of our approaches and explore their potential benefits. We investigate therefore examples with different inherent features (e.g. smooth and non-smooth behavior of the analytic solution) in order to get a qualified impression of their behavior. In the first part we study different post-processing techniques applied to 1D, 2D and 3D problems. We evaluate four different projection approaches derived and explained in the previous sections. In all examples shown we will systematically explore h- or/and k-mesh refinement. In the second part we describe visualization challenges exploiting the presented techniques and tools from Section 4 and show the flexibility of the proposed visualization technique suitable for structured and unstructured mesh representations, linear and non-linear elasticity results, 2D, 3D and cut plane visualization, multi-patch model –, time evolution – and immersed boundary method representations. But first we introduce certain measures (norms) for proper evaluation of our numerical test examples.

5.1. Evaluation measures

In the following we define some error/difference measures that we exploit during the evaluation of the different test examples. Let u be the exact solution and u^h the approximate finite element solution. The errors are denoted by

$$e(x) = u(x) - u^h(x) \tag{46}$$

$$e_\sigma(x) = \sigma(x) - \sigma^h(x) = CDe(x) \tag{47}$$

$$e_{\sigma^*}(x) = \sigma(x) - \sigma^*(x) \tag{48}$$

$$e_d(x) = \sigma^*(x) - \sigma^h(x) \tag{49}$$

where e denotes the error in the displacements u^h, e_σ the error in the stresses σ^h, e_{σ^*} the error in the recovered stresses σ^* and e_d the difference between the stresses σ^h and the recovered stresses σ^*. We use the following error norms

$$\|e_{()}\|^2_{L_2(\Omega)} = \int_\Omega e_{()}^\top e_{()} dV \tag{50}$$

$$\|e_{()}\|^2_{E(\Omega)} = \frac{1}{2}\int_\Omega (e_{()})^\top C^{-1} e_{()} dV, \tag{51}$$

where $e_{()}$ can be replaced by e_d, e_{σ^*} or e_σ. The error plots will show the relative difference in L_2-norm

$$\frac{\|e_d\|^2_{L_2(\Omega)}}{\|\sigma^*\|^2_{L_2(\Omega)}} = \frac{\|\sigma^* - \sigma^h\|^2_{L_2(\Omega)}}{\|\sigma^*\|^2_{L_2(\Omega)}}, \tag{52}$$

the relative difference in energy norm

$$\frac{\|e_d\|^2_{E(\Omega)}}{\|\sigma^*\|^2_{E(\Omega)}} = \frac{\|\sigma^* - \sigma^h\|^2_{E(\Omega)}}{\|\sigma^*\|^2_{E(\Omega)}}, \tag{53}$$

and the relative global error in L_2-norm

$$\frac{\|e_{\sigma^*}\|_{L_2(\Omega)}}{\|\sigma\|_{L_2(\Omega)}} \cdot 100\% = \frac{\|\sigma - \sigma^*\|_{L_2(\Omega)}}{\|\sigma\|_{L_2(\Omega)}} \cdot 100\%. \tag{54}$$

Table 1
1D-Smooth: Performance of the relative global $L2$-error for degrees $p = 1, 2, 3, 4$.

	Global methods			Local methods	
p	Interpl.	Greville	Least-Sq.	Quasi	VDSA
1	1.851%	1.851%	0.813%	1.851%	1.851%
2	0.231%	0.188%	0.150%	0.285%	3.216%
3	0.107%	0.090%	0.053%	0.297%	5.319%
4	0.032%	0.037%	0.027%	0.099%	7.411%

Table 2
1D-NonSmooth: Performance of the relative global L^2-error for degrees $p = 1, 2, 3, 4$.

	Global methods		Local methods	
p	Greville	Least-Sq.	Quasi.	VDSA
1	4.086%	3.896%	4.086%	4.086%
2	4.156%	4.107%	6.4244%	4.747%
3	4.686%	4.589%	14.067%	5.598%
4	5.076%	4.882%	34.416%	6.457%

5.2. Evaluation of the projection methods

We will in the following show results obtained with the four different projection techniques developed in Section 2 applied to numerical examples. Here we will provide a local content in order to give an overview of the different experiments.

5.2.1. 1D-Smooth

The results for the smooth test function are shown in Fig. 24 and Table 1. We report the *relative global error in L_2-norm* defined by (54) for each method using polynomial degree $p = \{1, 2, 3, 4\}$.

The performance for the global methods, with respect to the relative global error, leads to superior results compared with the presented local methods. We observe that spline interpolation using Greville points results in a slightly better fit than using equidistant interpolation points (cf. Table 1).

The least-squares method gives the best fit, i.e. lowest relative global error. The spline interpolation gets closer to the least-squares result for increasing polynomial degree. This is also the case for the quasi-interpolation method but the difference is larger (cf. Fig. 26 left). In contrast to that we observe, that for the local VDSA the relative global error increases with increasing degree.

5.2.2. 1D-NonSmooth

We observe also for this non-smooth test example shown in Fig. 25 that the global least-squares approximation method performs best (cf. Table 2 and Fig. 26 right), but both spline interpolation using Greville points and VDSA gives results of comparable accuracy. Using the presented local quasi-interpolation for this kind of problems seems not advisable, as for degree higher than three the discontinuity of the function cannot be represented.

5.2.3. Condition number

If we look at the condition numbers for the two global methods (Fig. 27) we observe the spline interpolation method performs uniformly and superior to the dense mass matrix of the least-squares method.

The reason for that lies in the sparsity of the matrix A of the spline interpolation problem of (9), whereas the mass matrix $N = A^\top A$ (12) is much more dense. As a consequence, in general, the least-squares method is more costly to solve than the better conditioned spline interpolation method.

Since the VDSA method exploits directly the derivative evaluation points as the control points, the creation of a higher-order derivative does not imply the solution of a linear system of equations and is therefore the least costly method as shown in Section 5.2.8 also for 2D examples.

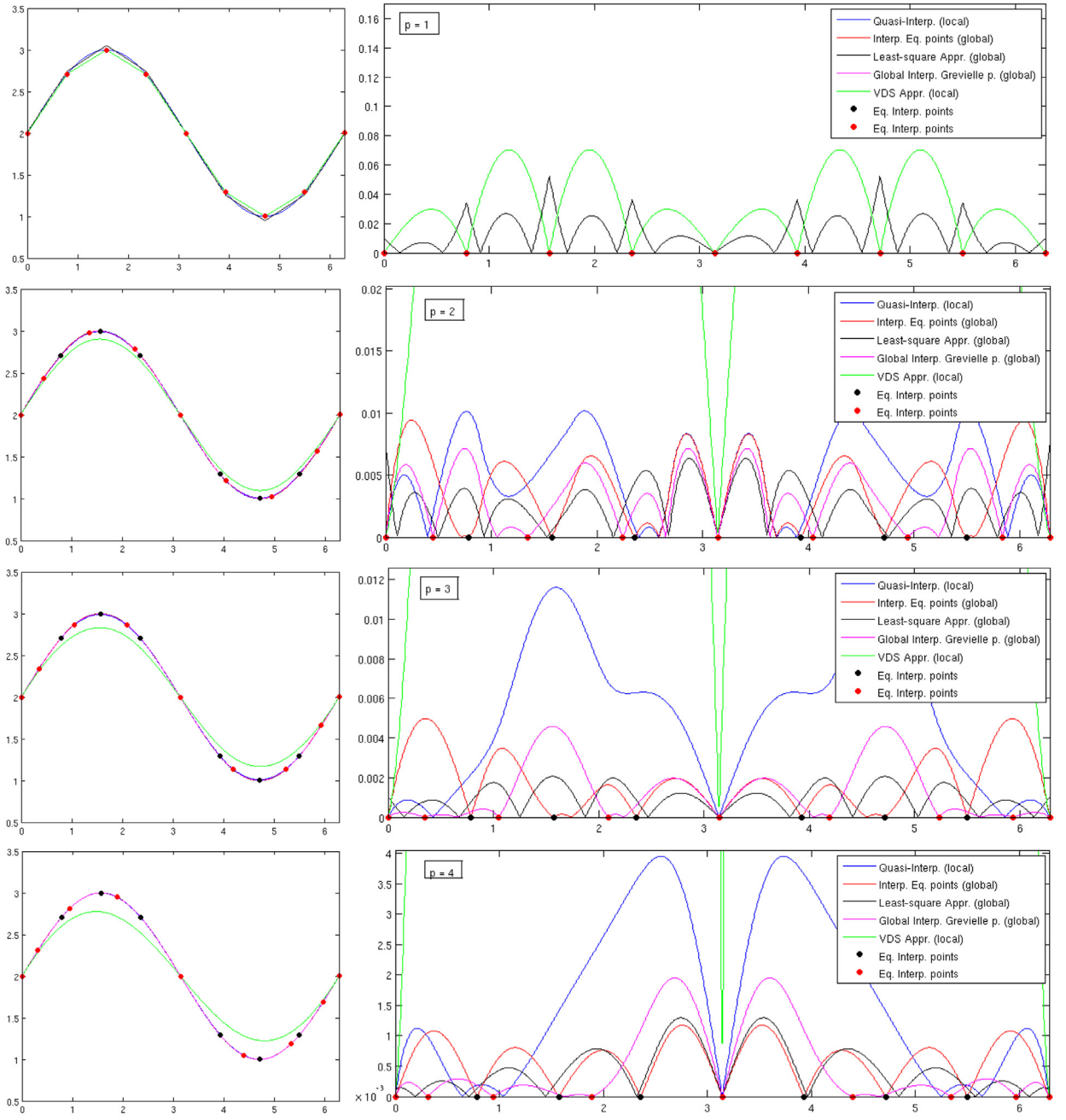

Fig. 24. 1D-Smooth: **Left:** Comparison between the exact $u = \sin(x) + 2$ function and the global methods: Spline interpolation evaluated at equidistant and at Greville interpolation points and least-squares approximation. As well as the local methods: Quasi-interpolation and VDSA. **Right:** L^2-errors of the corresponding methods.

5.2.4. 2D-Lshape

In this section we investigate the projection results for the scalar-valued von Mises stress over the well-known 2D L-shaped problem. The geometry is constructed as a cubic NURBS with knot vectors $\Xi = [00001112222]$, $\mathcal{H} = [00001111]$ so that only the endpoint in the η-parameter direction are included, while the ξ-direction has a C^0-continuity line in the midpoint of the parameter space allowing to design the sharp edge formation at the origin

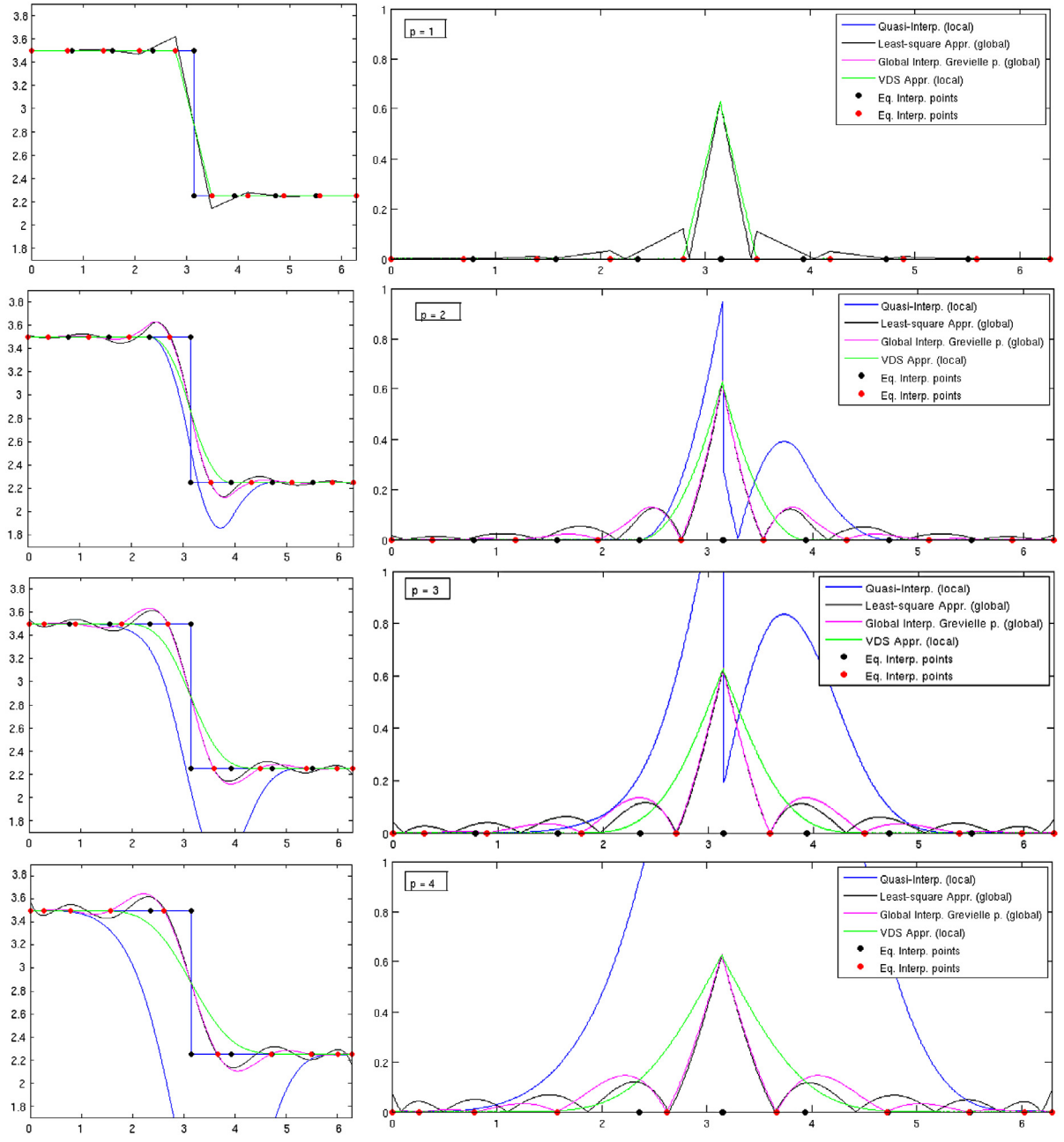

Fig. 25. 1D-NonSmooth: **Left:** Comparison between the exact discontinuous function and the global methods: Spline interpolation evaluated at Greville interpolation points and least-squares approximation. As well as the local methods: Quasi-interpolation and VDSA. **Right:** L^2-errors of the corresponding methods.

creating a singularity. This line will split the domain into two elements by the diagonal axis as indicated in the first image of Fig. 28.

The contour plots for the von Mises stress results are shown in Fig. 29, where we first show the analytic solution to this problem along with the result created in the one order lower displacement spline space. The projection results are plotted in the following order: Quasi-interpolation, VDSA, spline interpolation at Greville points and least-squares approximation. The respective difference measure using the energy norm is shown in Fig. 30 and in logarithmic representation in Fig. 31. The least-squares projection method shows a uniform difference distribution. The VDSA

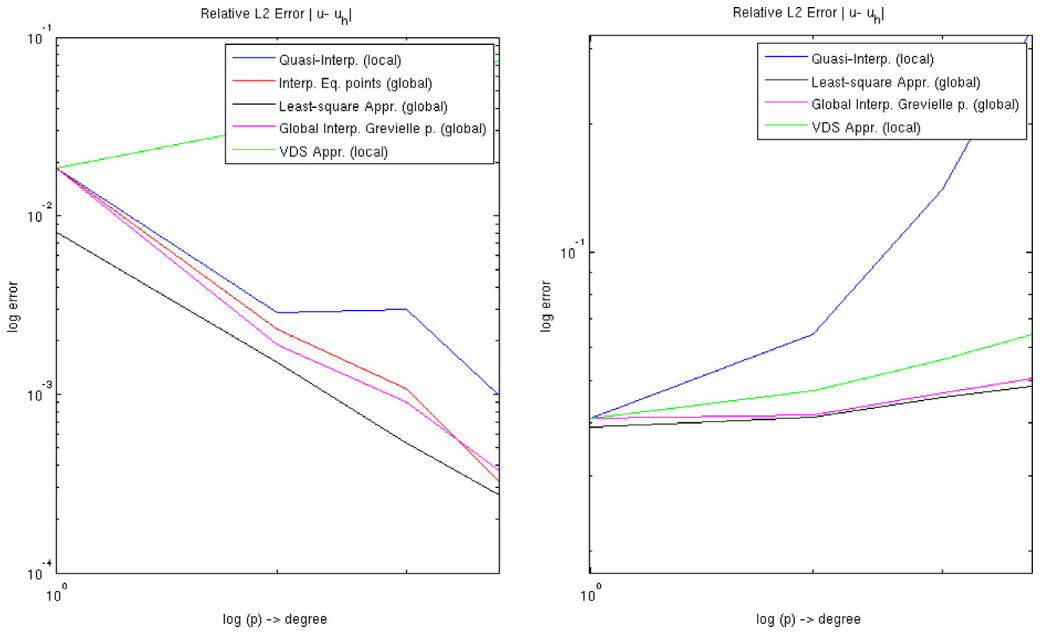

Fig. 26. **Left:** 1D-Smooth: Relative global L^2-error (see also Table 1). **Right:** 1D-NonSmooth: Relative global L2-error (see also Table 2).

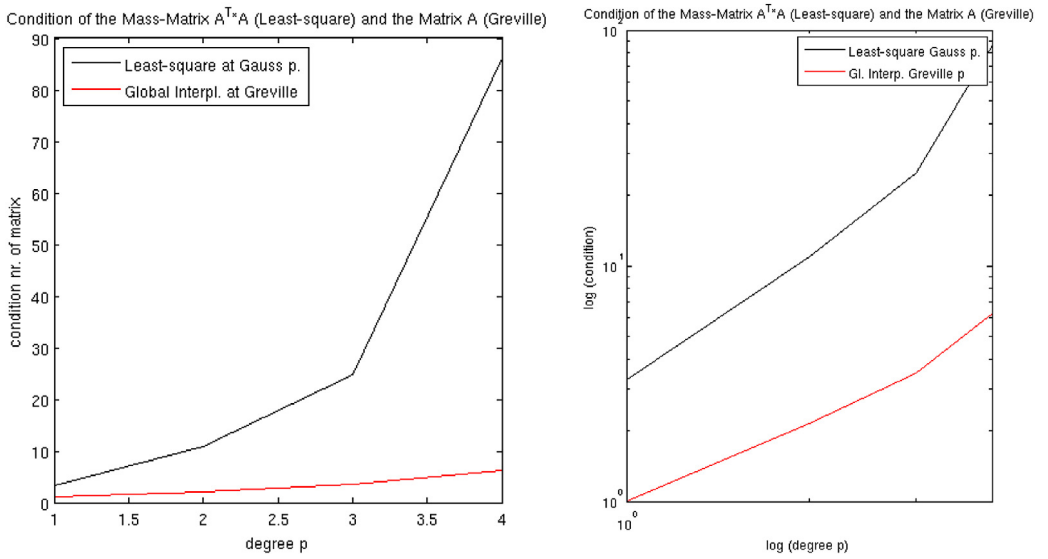

Fig. 27. 1D-Smooth: **Left:** Matrix condition number for the least-squares fit and the global spline interpolation method at Greville points (see also Table 1). **Right:** Max-norm error.

projection method not relying on the solution of a linear system of equations, with shape preserving properties produces a more or less homogeneous result except the singularity point at the origin.

Comparing the convergence plots in Figs. 32 and 33 we clearly see that the least-squares projection method performs best in all cases. As already experienced in the one-dimensional cases the interpolation at Greville points gives also good results. And both local methods are not as well suited in this example as the global approaches.

5.2.5. 3D-Hole (corner singularity)

In this section we discuss the three-dimensional version of the well known circular hole problem [17,31]. The quantity of interest is as in the previous section the scalar von Mises stress field as stated in Eq. (6). The exact

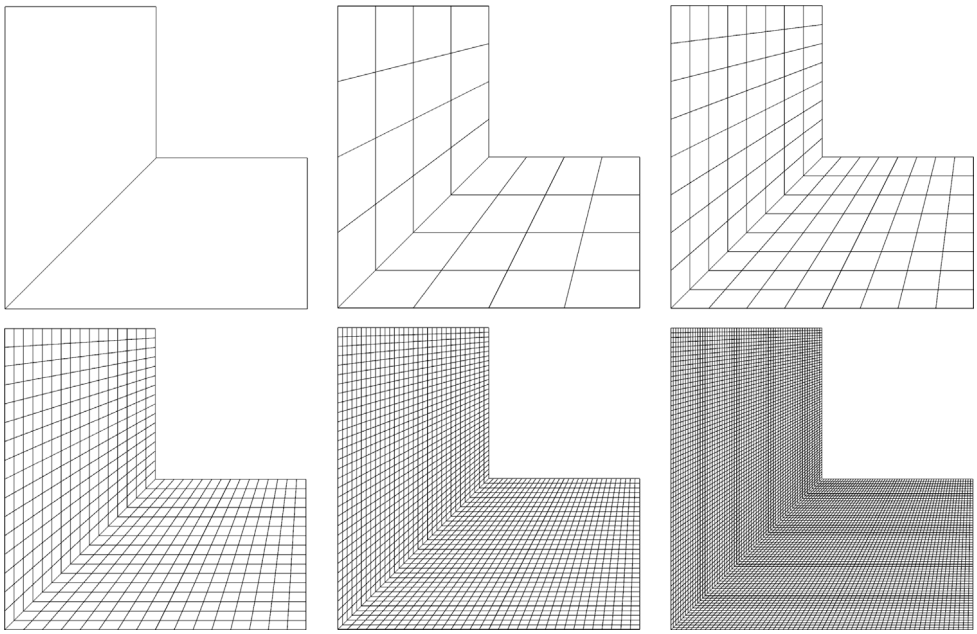

Fig. 28. 2D-Lshape: Uniform mesh refinements. From 0, 3, 7, 15, 31 and 61 interior knots per element.

geometry is represented with two elements as shown in Fig. 34 and defined by the mesh $\Xi \times \mathcal{H} \times \Theta$, with knot vectors $\Xi = [0001222]$, $\mathcal{H} = [000111]$ and $\Theta = [0011]$. In the upper right the coalescence of two control points implies singularities in the inverse of the geometrical mapping at the corners.

Using the least-squares approach exploiting Gaussian quadrature points the singularity at the upper right corners does not create problems in the analysis (cf. Fig. 37), whereas in the case of the other three projection approaches interpolation points at the location of the singularity leads to less-accurate approximation and projection results as shown in Fig. 37 as well as in the convergence plots of Figs. 38 and 39.

Using knot insertion we obtain the six meshes (shown in Fig. 34) which we used in the analysis. The respective contour plots of the analysis results obtained on the mesh with 31 interior points are shown in Fig. 35. We show in the first row the analytic solution and the von Mises stress in the spline space one order lower than the displacement field. The results of the four projection methods are shown in the second row (quasi-interpolation and VDSA) and in the third row the global projection based on spline interpolation and least-squares approximation. The respective relative difference measure is shown in Fig. 36.

Convergence results in L^2-norm and energy norm are shown in Figs. 38 and 39 respectively. For polynomial degree $p = 3, 4, 5$ results are obtained by order elevation of the quadratic NURBS on the coarsest mesh. Knot insertion generates identical meshes for all degrees. It can be seen that the convergence rates for this 3D example behave in the same way as in the 1D and 2D example, where the least-squares projection method gives the best result and higher convergence rates for higher degrees, where the remaining three methods are mostly in the same convergence rate class.

5.2.6. 3D-Hole (C^0-line)

We use the same setup with a different parametrization for the circular hole problem in 3D. We perform a C^0-parametrization and defined the mesh $\Xi \times \mathcal{H} \times \Theta$ as shown in Fig. 40 with the knot vectors $\Xi = [00011222]$, $\mathcal{H} = [000111]$ and $\Theta = [0011]$. That means we produce a C^0-line within the geometry between the two elements. With knot insertion we obtain six meshes as previous with 0, 3, 7, 15, 31 and 61 interior knots.

With this parametrization we have no high error measurements at the upper right corner as in the previous example where a singularity for the spline interpolation, VDSA and quasi-interpolation method created difficulties in the approximation which had implications for the resulting convergence plots. If we look at the convergence plots for the C^0-parametrization in Figs. 43 and 44 we obtain a much more balanced situation for this parametrization where no singularity occurs. This is also observable in the contour plots of the von Mises stress results in Fig. 41 and the

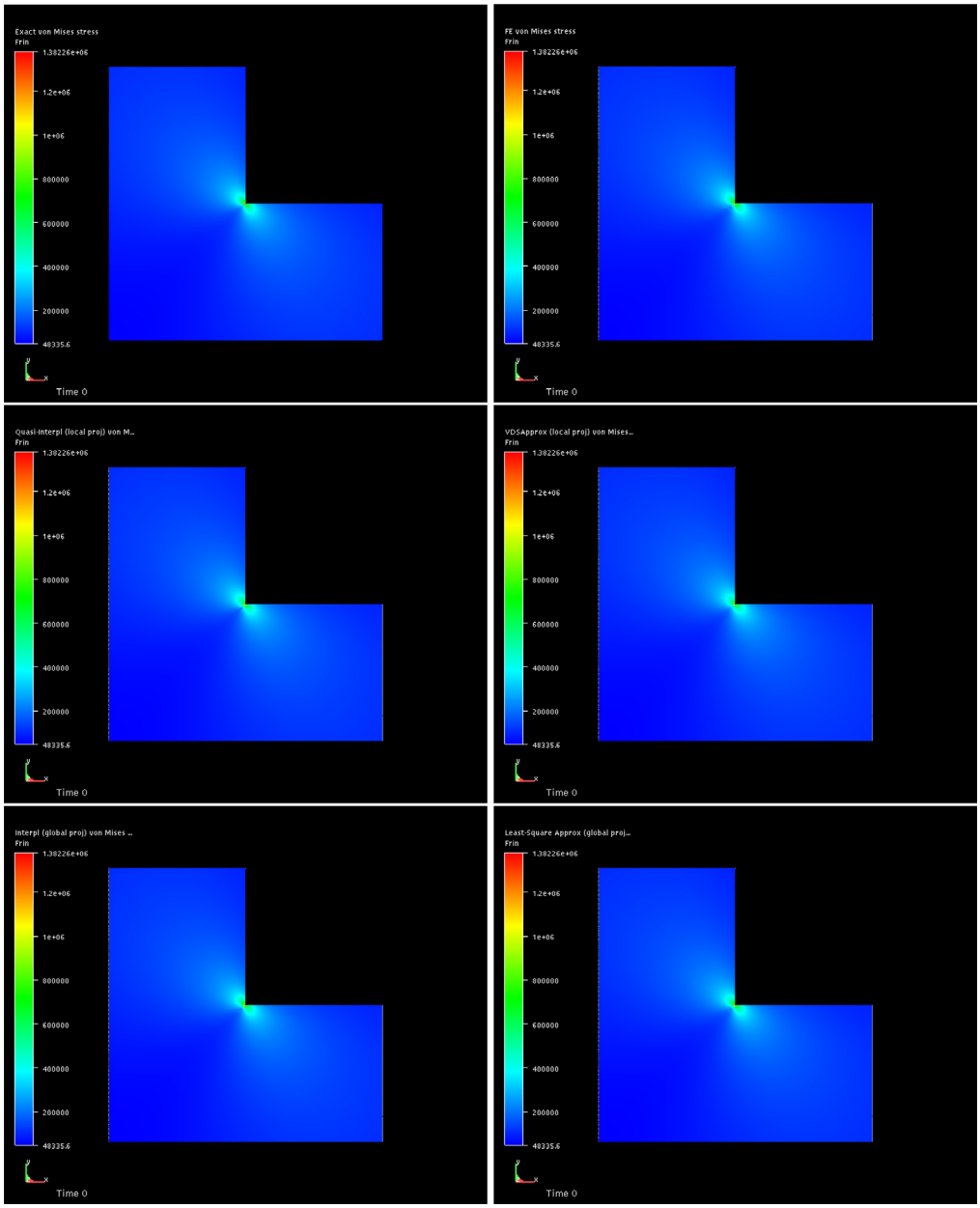

Fig. 29. 2D-Lshape: Displaying von Mises stress. **Top left:** Exact solution. **Top right:** Direct finite element solution. **Middle left:** Quasi-interpolation projection method. **Middle right:** VDSA projection method. **Bottom left:** Projection method exploiting spline-interpolation at Greville points. **Bottom right:** Least-squares projection approach.

error plot of Fig. 42. The exception is the VDSA which produces, due to the shape preserving properties, a smoothing across the C^0-continuity and leads therefore to a less accurate fit. Also the quasi-interpolation approach produces problems across the C^0-continuity line as already observed in the one dimensional case setup.

5.2.7. 3D-Hole (2 patches)

To investigate the previous circular hole examples even further we test a two patch representation where the mesh is defined by $\Xi \times \mathcal{H} \times \Theta$, with knot vectors $\Xi = [000111222]$, $\mathcal{H} = [000111]$ and $\Theta = [0011]$ as shown in Fig. 45.

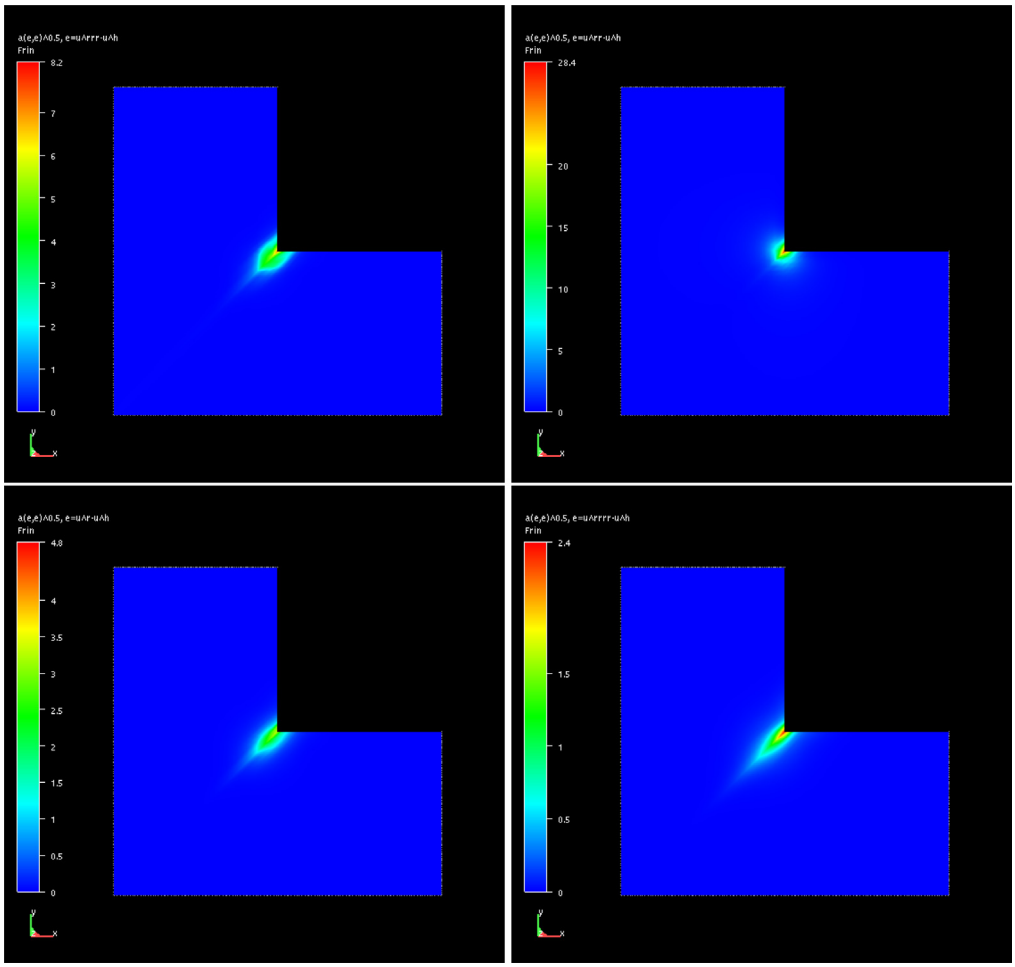

Fig. 30. 2D-Lshape: Displaying relative difference in the energy norm. **Top left:** Quasi-interpolation projection method. **Top right:** VDSA projection. **Bottom left:** Projection method exploiting spline-interpolation at Greville points. **Bottom right:** Least-squares projection approach.

The contour plots for this example are shown in Fig. 46. By removing also the C^0-continuity line and forming two patches we obtain for the quasi-interpolation approach a better result, whereas the VDSA has even more problems around the patch boundaries of C^{-1}-continuity. The convergence plots for this example are shown in Figs. 47 and 48.

5.2.8. *Time performance*

In this section we study the time performance of each of the four presented methods as shown in Fig. 49. The conditioning number of the coefficient matrix for the projection method exploiting spline interpolation at Greville points is lower than the conditioning number of the coefficient matrix for the least-squares approach and this is the reason for the difference in the time performance for solving the respective system for these two global methods. Due to the fact that no linear system of equations is solved in the VDSA method the performance with respect to the computational cost is superior in comparison to all the other methods. The local quasi-interpolation approach which involves small equation systems has a time consumption in between the global methods and the VDSA method.

5.2.9. *Discussion of preferred projection technique*

The result of our in depth studies of the four projection methods may be summarized as follows:

- The global least-squares projection is the most accurate and the most time consuming one.
- The VDSA is the less accurate but most efficient one.

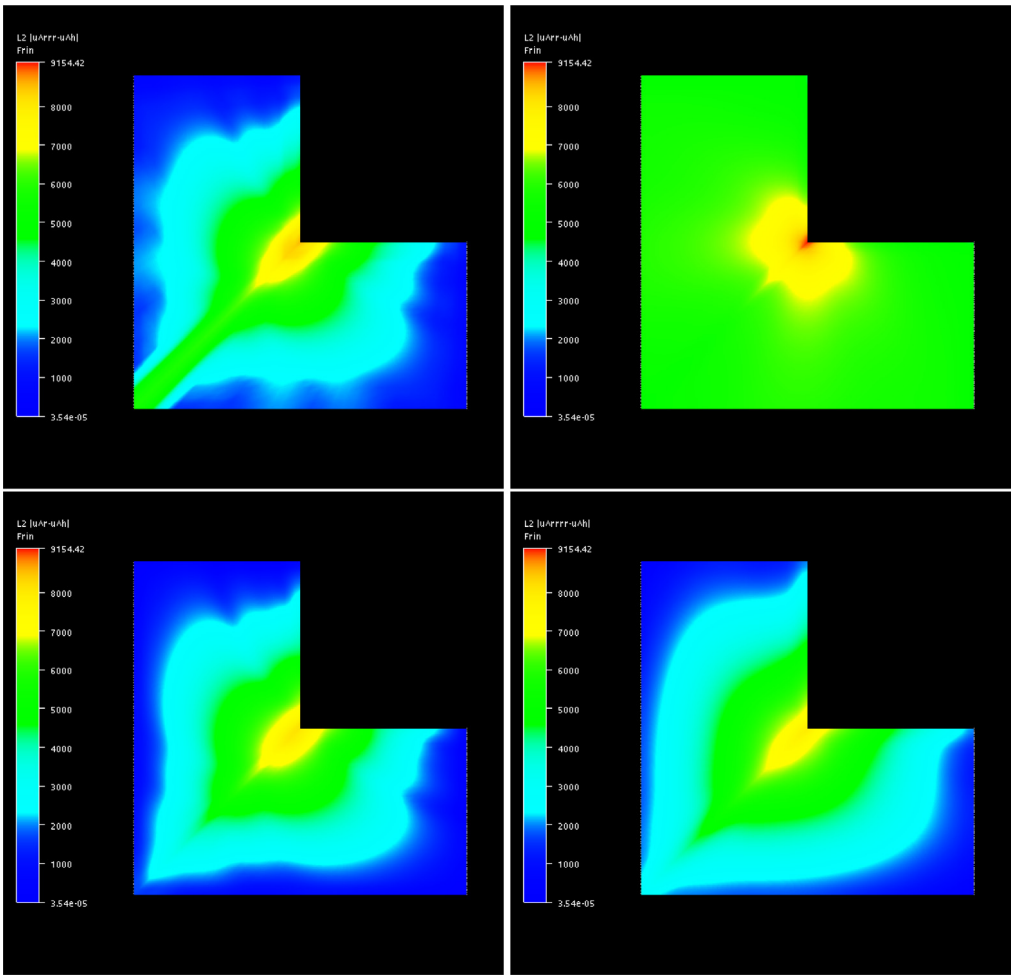

Fig. 31. 2D-Lshape: Displaying relative difference in the energy norm (logarithmic). **Top left:** Projection method exploiting quasi-interpolation. **Top right:** VDSA projection method. **Bottom left:** Projection method exploiting spline-interpolation at Greville points. **Bottom right:** Least-squares approximation projection.

- The global least-squares projection produces reliable results for all the tested cases.
- The quasi-interpolation, VDSA and the Greville point projection fails to provide reliable results in some of the tested cases.

Based on this investigation we will propose to use the global least-squares projection for small to medium sized problems, as the extra computational time should not be of any concern. However, for large scale problems the post-processing should preferably grow linearly with the number of unknowns (which is equivalent to a constant times the number of elements) and this is indeed not the case for the global methods. Thus, we will therefore mention the work by Kumar, Kvamsdal and Johannessen presented in [32] where they have developed a Superconvergent Patch Recovery (SPR) technique for isogeometric methods. Here they solve for each basis function local least-squares problems defined on reasonable small patches of elements. They compared the IGA-SPR procedure with the global least-squares projection and the IGA-SPR gives equal or better accuracy. Another option may be the local least-squares method proposed in [33] where the error is bounded by a constant times the error for the global least-squares method.

5.3. Pixel-accurate spline visualization of engineering problems

After the post-processing, i.e. projection, we have a convenient data structure at hand. The basis function set for the secondary analysis results are represented in the same function space as the primary results and structured or unstructured mesh representations are converted to a Bézier element representation allowing for parallel processing on GPU.

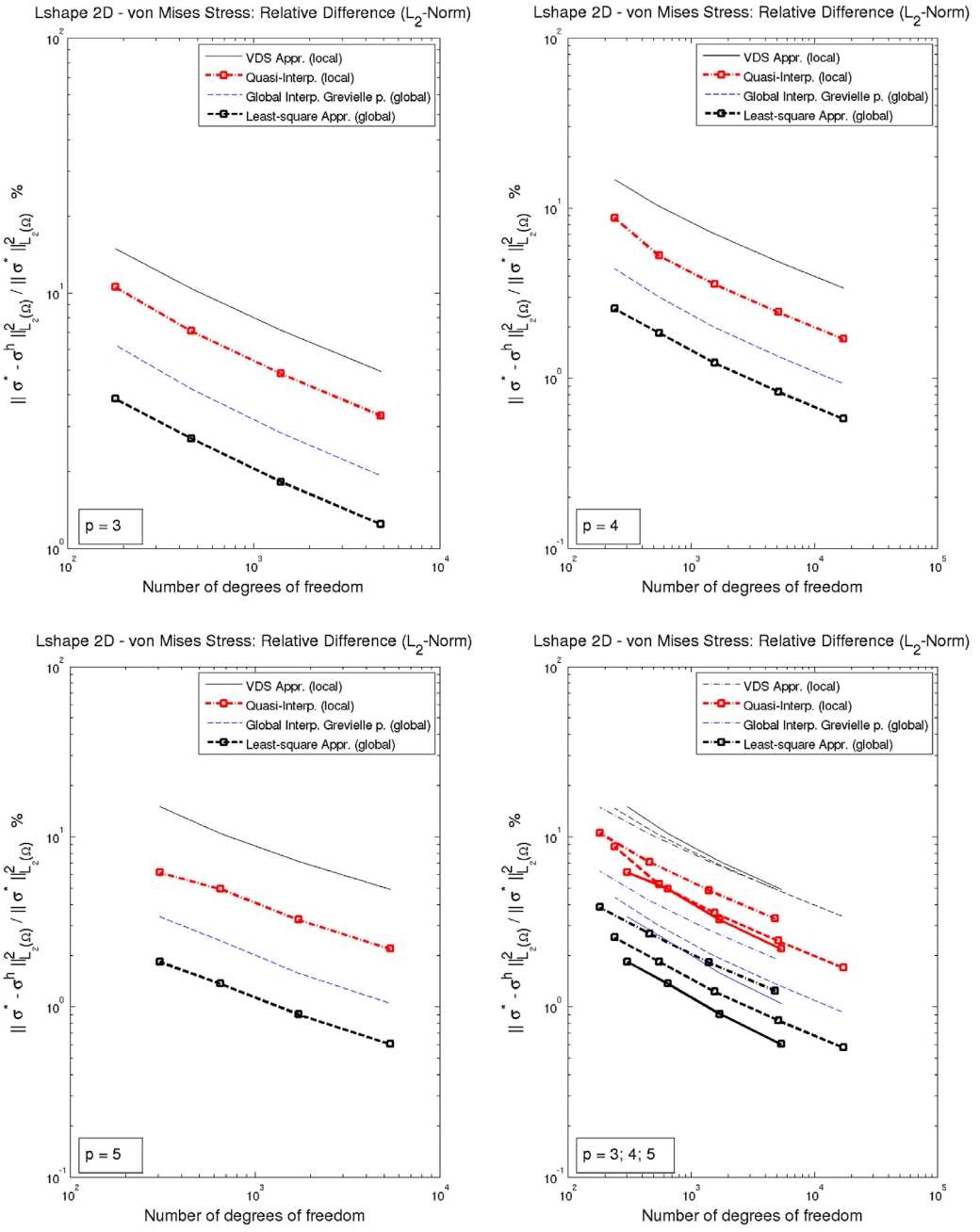

Fig. 32. 2D-Lshape: Displaying relative difference (L_2-norm) of von Mises stresses: $\|\sigma^* - \sigma^h\|^2_{L_2(\Omega)} / \|\sigma^*\|^2_{L_2(\Omega)}$.

For all the numerical examples shown in this section this data structure is exploited. We show visualization results using techniques for trimming, cutting planes, error element view and time evaluation of B-splines, NURBS and adaptive LR B-splines.

5.3.1. Visualization comparison: Finite element tessellation — Pixel-accurate splines

In this section we show the visualization outcome of conventional methods using linear finite element meshes as well as pixel-accurate spline evaluation for visualization proposes. We have chosen an adaptive local refinement method solving the Poisson problem for the two dimensional Lshape. The adaptive method is defined by refining

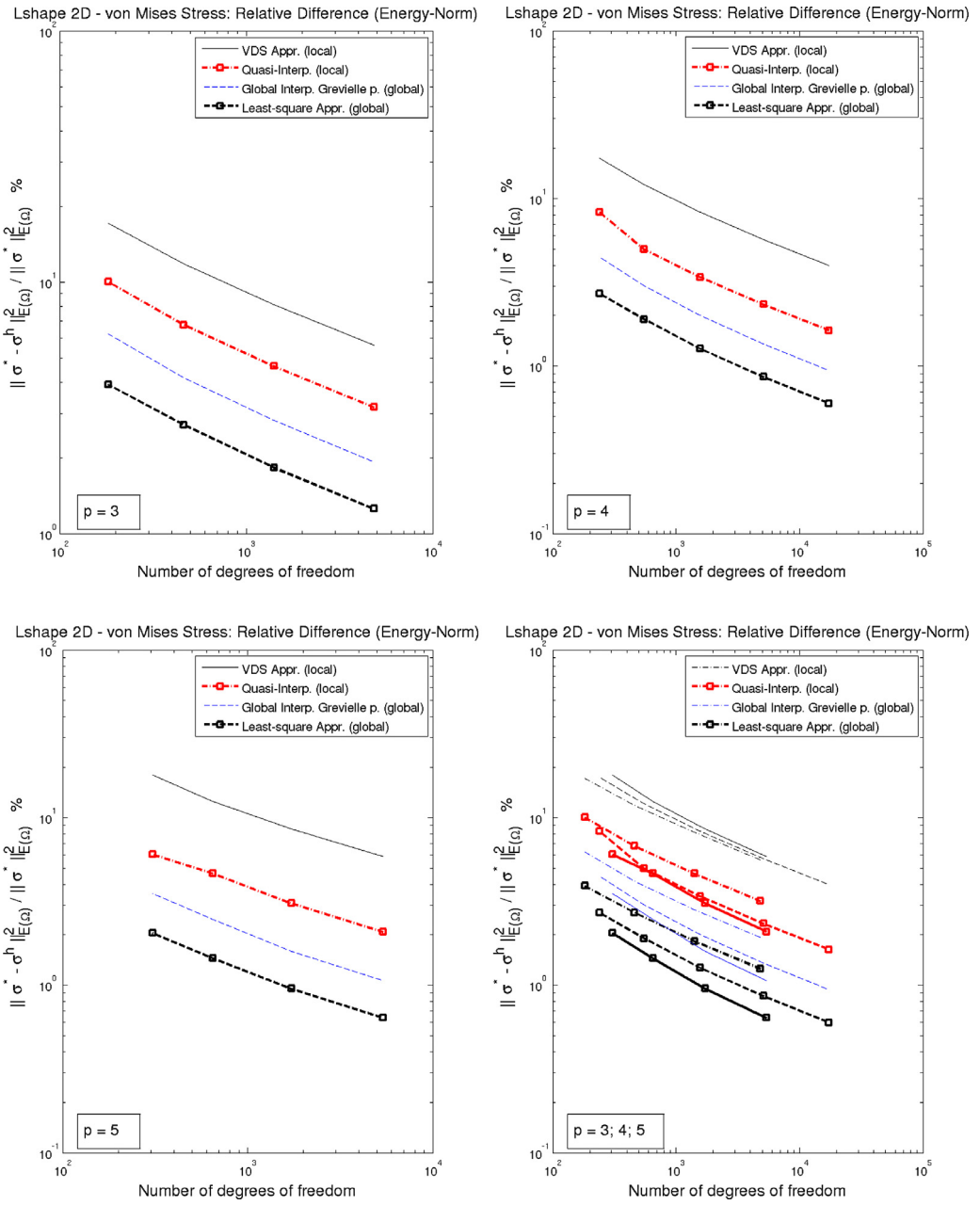

Fig. 33. 2D-Lshape: Displaying relative difference (energy norm) of von Mises stresses: $\|\sigma^* - \sigma^h\|^2_{E(\Omega)} / \|\sigma^*\|^2_{E(\Omega)}$.

the support of the basis function with 10% highest percentage energy error (see [3]). Here, the adaptive sequence is stopped after 10 refinements. In Fig. 50 the obtained solution of the Poisson problem is shown for the refinement level 9. Refinement is mostly performed around the singularity point of the Lshape. Thus, the most interesting area for displaying the obtained mesh and computed von Mises stresses is in the vicinity of the singularity point. In Fig. 51 we have therefore in the center and right image shown a zoomed version of the von Mises stresses visualized using linear finite element tessellation and pixel-accurate spline evaluation, respectively. With pixel-accurate rendering we can shown on every zoom level accurate computation results, that was not possible with the previously used finite element tessellation approaches as demonstrated here.

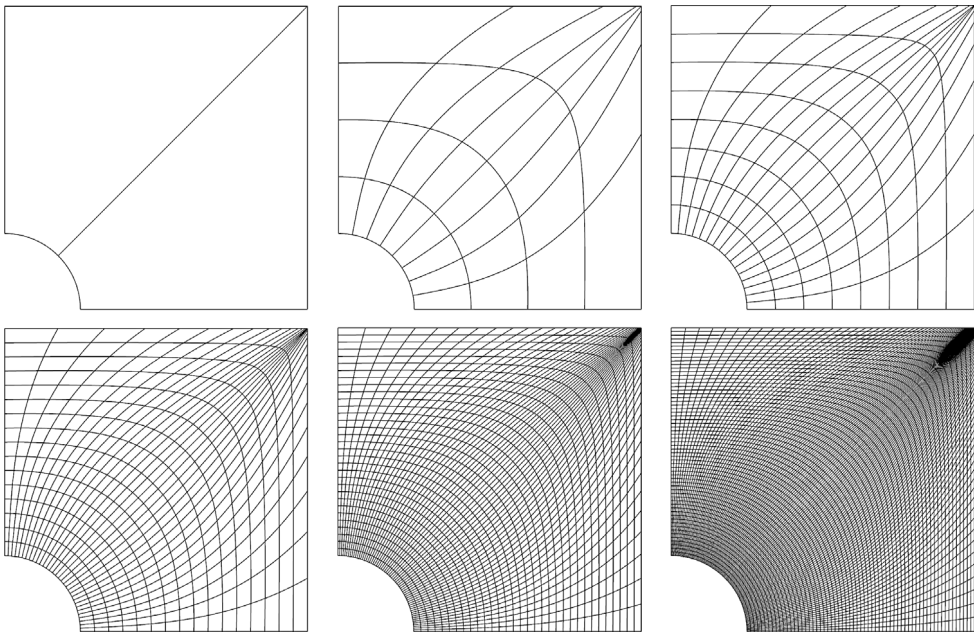

Fig. 34. 3D-Hole (corner singularity): Uniform mesh refinements. From 0, 3, 7, 15, 31 and 61 interior knots per element.

5.3.2. *Volume-element-view for 3D LR B-splines*

Exploiting the pixel-accurate volume rendering technique for 3D LR B-spline analysis, results can be shown in a volume-element-view allowing a better understanding of the refined regions. In Figs. 52 and 53 two different adaptive mesh refinements are displayed for 5 refinement stages, respectively.

Volume elements can be systematically removed and visualized in different ordering allowing also better insights in analysis error plots (cf. Figs. 53 and 54).

5.3.3. *Cutting planes*

Data sets in scientific visualization often take the form of 3D scalar fields as shown in this work. Scalar fields, which are especially difficult to understand both because they are not easily rendered on a 2D screen and because they often contain enormous amounts of data require special visualization representations. One informative way of visualizing details of a scalar field representation utilizing volume rendering are cut plane or isosurface rendering.

A 3D equivalent of a single contour line is an isosurface providing a continuous surface that passes through all voxels containing the specified value of the scalar field (as described in Section 4.1). Isosurface rendering provides in general a more global view of a particular value in the field than cut planes, but do not illustrate how the data changes during the volume and in addition to that it has higher computational costs. Cut plane rendering on the other hand samples a 3D scalar field along a particular geometrical plane, mostly a planar 2D plane (slice) through the data set. Slice or cut plane rendering allows the user to view a manageable subset of the data set and to explore the field by sweeping the plane through the 3D volume space.

Exploiting the ray casting approach we are able to render cutting planes through a LR B-spline volume as shown in Fig. 55 by computing the intersection of the ray with a particular cut plane of the volume. The respective scalar field value at the particular 2D position (since one dimension is set to zero) is then evaluated using the approach described in Sections 4.2 and 4.2.4.

5.3.4. *Multi-patch models: Linear elasticity*

In order to model geometrical entities with a certain complexity it is often necessary to utilize multi-patch modeling. As a consequence, multiple patches have to be visualized simultaneously. Patches are evaluated in parallel during the visualization process. Linear elasticity results are shown in Figs. 56 and 57, where nodal averaging is done between the patch boundaries.

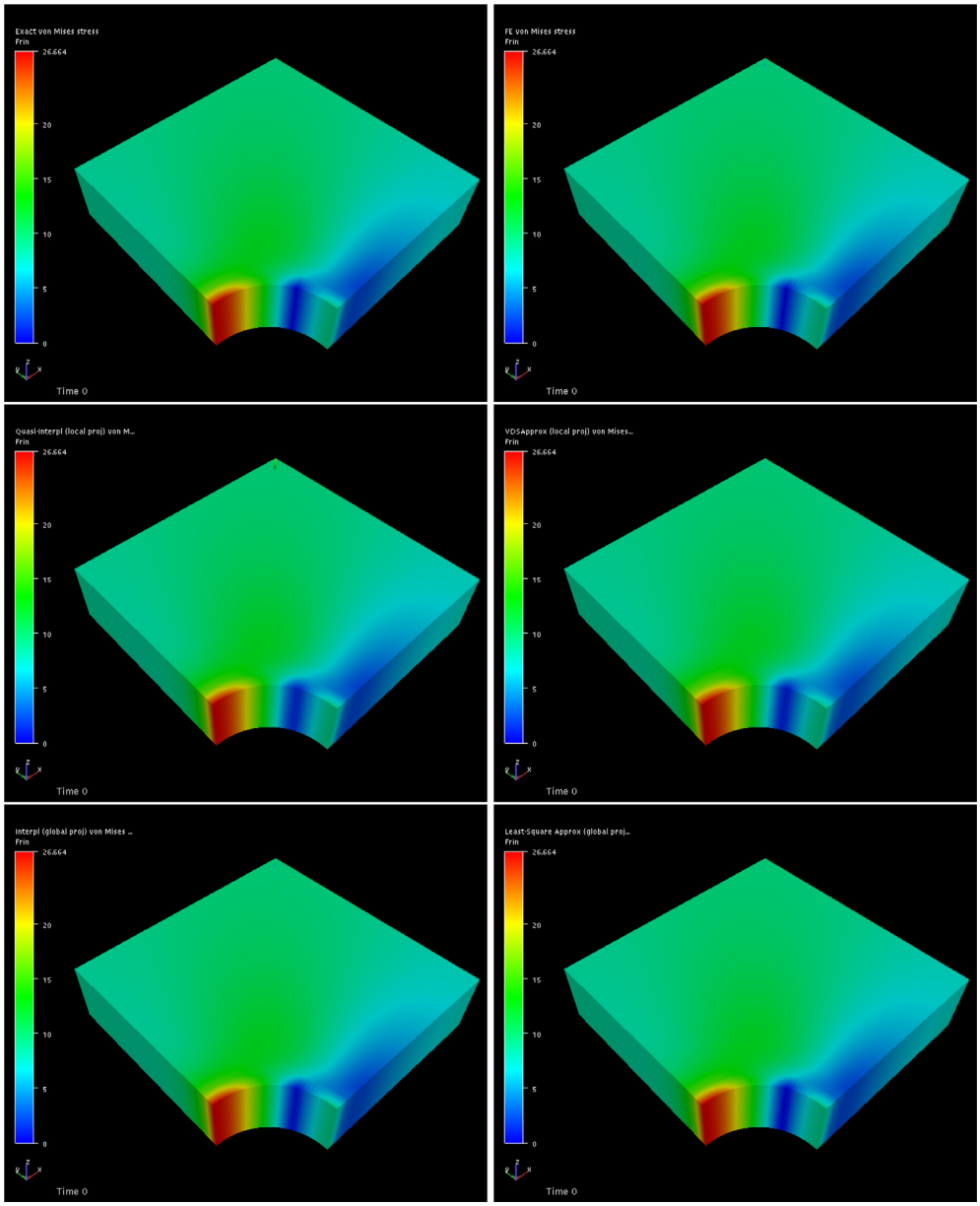

Fig. 35. 3D-Hole (corner singularity): Displaying von Mises stresses. **Top left:** Exact solution. **Top right:** Direct finite element solution. **Middle left:** Projection method exploiting quasi-interpolation. **Middle right:** VDSA projection. **Bottom left:** Projection method exploiting spline-interpolation at Greville points. **Bottom right:** Projection exploiting least-squares approximation.

5.3.5. Time evolutions: Non-linear elasticity

In this section we present pixel-accurate rendering results regarding time evolving non-linear elasticity solutions using the non-linear material models in IFEM [34]. The first test case displays the von Mises stress evolution of a subsea pipeline in contact with a trawl gear (herein denoted pipe contact), see Fig. 58. The simulations performed by IFEM [35] compares well with the experimental results presented in [36–38].

The second test case consists of a non-linear torsional deformation of a bar with resulting von Mises stresses proposed by [39].

A comparison between two visualization techniques are depicted in Fig. 59, where on the left-hand side a common tessellation method was exploited to mimic finite element meshing for visualization purposes, where on the right-hand

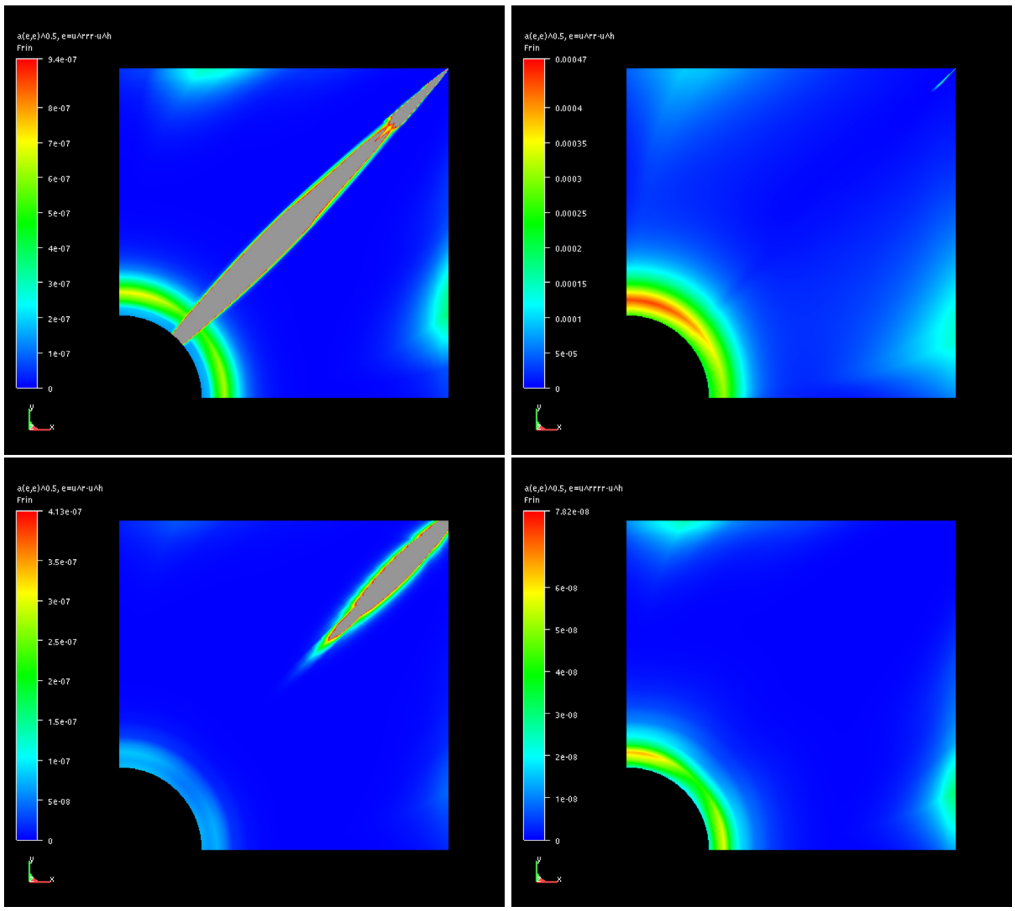

Fig. 36. 3D-Hole (corner singularity): Displaying relative difference of von Mises stresses measured in energy norm. **Top left:** Projection exploiting quasi-interpolation. **Top right:** VDSA projection. **Bottom left:** Projection method exploiting spline-interpolation at Greville points. **Bottom right:** Projection exploiting least-squares approximation.

side a pixel-accurate rendering is shown utilizing the proposed ray casting method. In Fig. 60 the resulting per-pixel evaluated bar is shown for 4 different time steps.

5.3.6. Trimming — Immersed boundary methods

Trimming describes the process of removing parts of a surface patch. One or more intersecting curves (trim curves) are predefined either in the parametric domain or in the physical domain. Trim curves are most likely defined as NURBS curves defining the region which has to be trimmed away.

Methods for immersed boundary problems consist of the extension of the physical domain of interest Ω_{phy} beyond its physical boundaries into a larger embedding domain of simple geometry Ω, which can be meshed easily by a structured grid [7]. Applying analysis on this grid structure means that the fictitious domain Ω_{fict} is extinguished by penalizing parameters.

This concept is applied to the circular hole problem as shown in Figs. 61 and 62 where the region $\Omega = \Omega_{phy} + \Omega_{fict}$ in the left images are shown. Analysis results within the physical domain are relevant and should be visualized. One possible solution for this is to trim the fictitious domain away using the shader programming ability in the visualization pipeline. The pixel set in the fictitious domain is defined lying in the trimming region marked by the trimming curve.

The typical problem which rises with this approach is the classification of points that lay in the physical or in the trimming domain. In order to solve this problem we exploit a special case of the point location problems denoted in computational geometry as the point-in-polygon (PIP) problem [27]. The domain within the trimming region is defined and the information is transferred to the fragment shader, where the individual pixel in that set is set to zero.

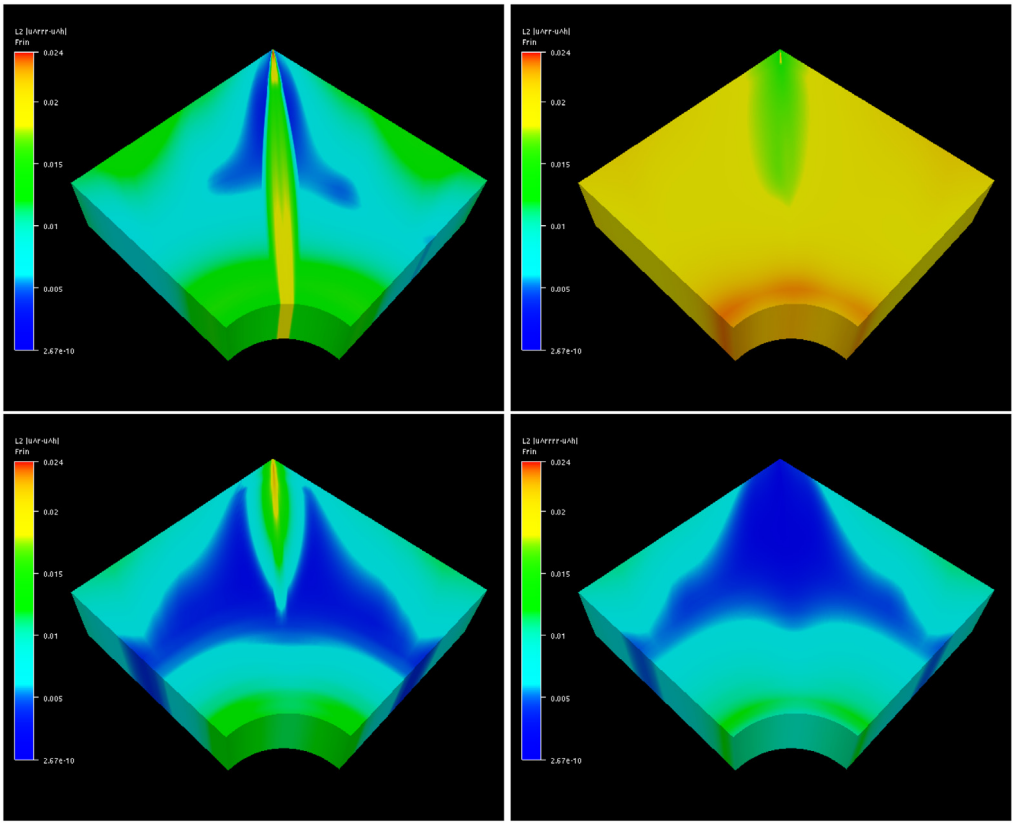

Fig. 37. 3D-Hole (corner singularity): Displaying the logarithmic of the relative difference of von Mises stresses in energy norm. **Top left:** Projection method exploiting spline-interpolation at Greville points. **Top right:** VDSA projection method. **Bottom left:** Projection method exploiting quasi-interpolation. **Bottom right:** Projection exploiting least-squares approximation.

Note, that for all the analysis results shown in this section we used the least-squares (L^2) projection approach as described and evaluated in the previous sections above.

In the next test example case we apply the method of adaptive refinement during the immersed boundary problem exploiting LR B-splines. The test case setup contains a simple supported rectangular plate (cf. Fig. 63), which is subjected to uniform pressure [40]. The fictitious domain contains of three circular holes which are trimmed away during the visualization process. The adaptive LR B-spline refinement technique [3] refines around the circular holes where Dirichlet boundary conditions are incorporated. The contour plots of the result for cubic spline Kirchhoff–Love thin plate elements (moment m_{xx}) are shown in Fig. 63 after the sixth refinement level.

For the last example in this section we show the model for an application in marine ship technology. The midship section was modeled with boundary fitted outer geometry whereas the holes where handled with the immersed boundary method [41] (cf. Fig. 64). This made the geometry modeling very easy and the resulting grid analysis suitable without the need for modification other than uniform refinement to achieve the desired accuracy. In the visualization process the fictitious region are trimmed away using the fragment processor ability (cf. Fig. 65).

6. Conclusion

In this paper we have developed and investigated an *accurate, efficient and practical post-processing pipeline for visualization of isogeometric analysis results*. The proposed IGA visualization pipeline consists of three steps: (1) *Projection*, (2) *Bézier decomposition* and (3) *Pixel-accurate rendering*:

1. Projection and representation of secondary analysis results in the same spline space as the primary results allows for an efficient and convenient data structure.

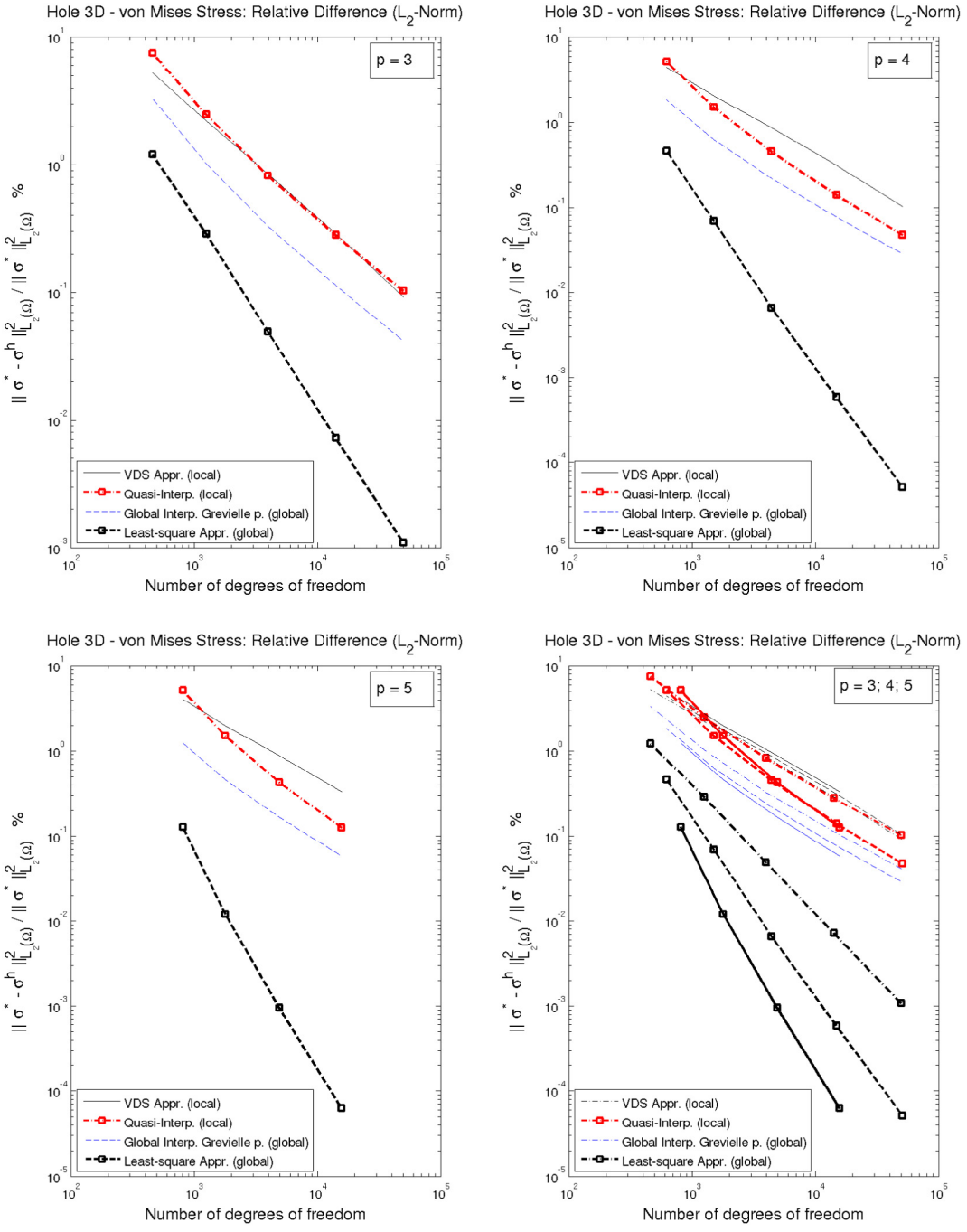

Fig. 38. 3D-Hole (corner singularity): Displaying the relative differences of von Mises stresses measured in L_2-norm: $\|\sigma^* - \sigma^h\|^2_{L_2(\Omega)} / \|\sigma^*\|^2_{L_2(\Omega)}$.

2. Bézier decomposition constitutes a common representation of structured and unstructured meshes like hierarchical splines, LR B-splines and T-splines techniques that also facilitates parallel processing during the visualization process.
3. Pixel-accurate rendering of spline basis evaluation on GPUs results in high quality visualizations of the isogeometric analysis results.

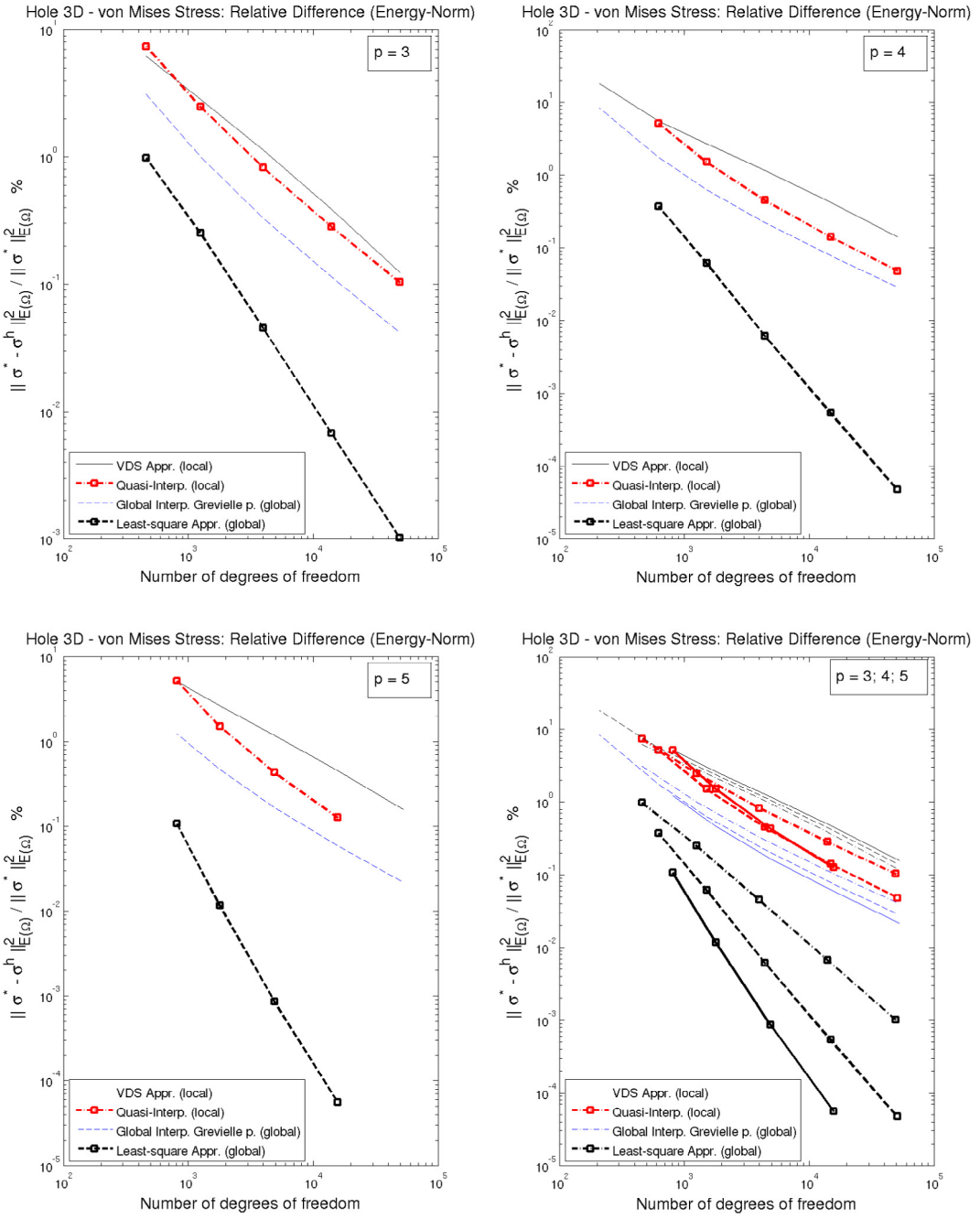

Fig. 39. 3D-Hole (corner singularity): Displaying the relative differences of von Mises stresses measured in energy norm. $\|\sigma^* - \sigma^h\|^2_{E(\Omega)}/\|\sigma^*\|^2_{E(\Omega)}$.

We compared four different approximation and interpolation methods with the aim to integrate the most suitable one into the projection scheme described above. The global least-squares fit is the best method for projection proposed here and provides optimal accuracy. The three other methods that has been tested were VDSA, quasi-interpolation and the global Greville point interpolation. They all failed to provide satisfactory accuracy in some of the tested cases. For small and medium sized problems the incurred cost by the global least-squares method is of no real concern and it is used by pioneers in the IGA community for large problems as well. However, for truly large scale problems we

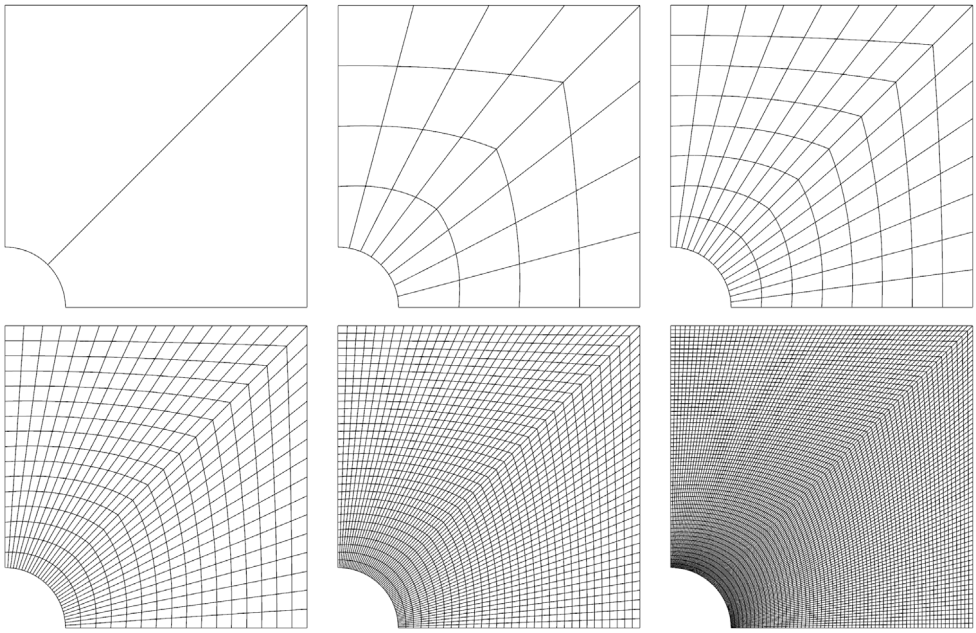

Fig. 40. 3D-Hole (C^0-line): Displaying uniform mesh refinements. From 0, 3, 7, 15, 31 and 61 interior knots per element.

propose the use of the IGA-Superconvergent Patch Recovery (IGA-SPR) as described in [32] or the local least-squares method presented in [33]. The IGA-SPR gives equal or better accuracy than the global least-squares method, whereas the error for the local least-squares method is bounded by a constant times the error for the global one. We showed that Bézier decomposition can be exploited to create an efficient data structure for structured and unstructured mesh topologies. We have described how to perform Bézier decomposition for LR B-splines, whereas Bézier decomposition for T-splines and hierarchical splines may be found in [9] and [6,7]. Thus, the IGA visualization pipeline developed herein is general in the sense that it may handle both global tensorial splines as well as locally refined splines. Visualization techniques like the described ray based method provide high performance with respect to image quality and computation time. Practical improvements of the method can be obtained by integrating the approach into standard and hierarchical data structures like octrees or exploiting early ray termination or initial value settings for the Newton method as described in [27]. We recommend to exploit also new developments appearing in graphics hardware leading to new programming techniques on GPUs. Therefore, frequent investigations and evaluations of new methods and developments, as done in [42] are required.

Acknowledgments

The authors acknowledge the financial support from the Norwegian Research Council and the industrial partners of the ICADA-project (NFR Project 187993); Ceetron, Det Norske Veritas and Statoil. They also acknowledge the support from the other co-workers in the ICADA-project. The authors thank in particular Kjetil Johannessen and Mukesh Kumar for many fruitful discussions and suggestions that have improved the manuscript.

Appendix

The concept for post-processing and visualization of isogeometric analysis proposed herein relies on the use of Bézier decomposition. To make the paper complete, regarding the visualization of IGA results obtained by use of LR B-splines, we present here in this Appendix the basic techniques of locally refined B-splines. But first we introduce basic definitions and notation used for the projection methods derived in Section 2.

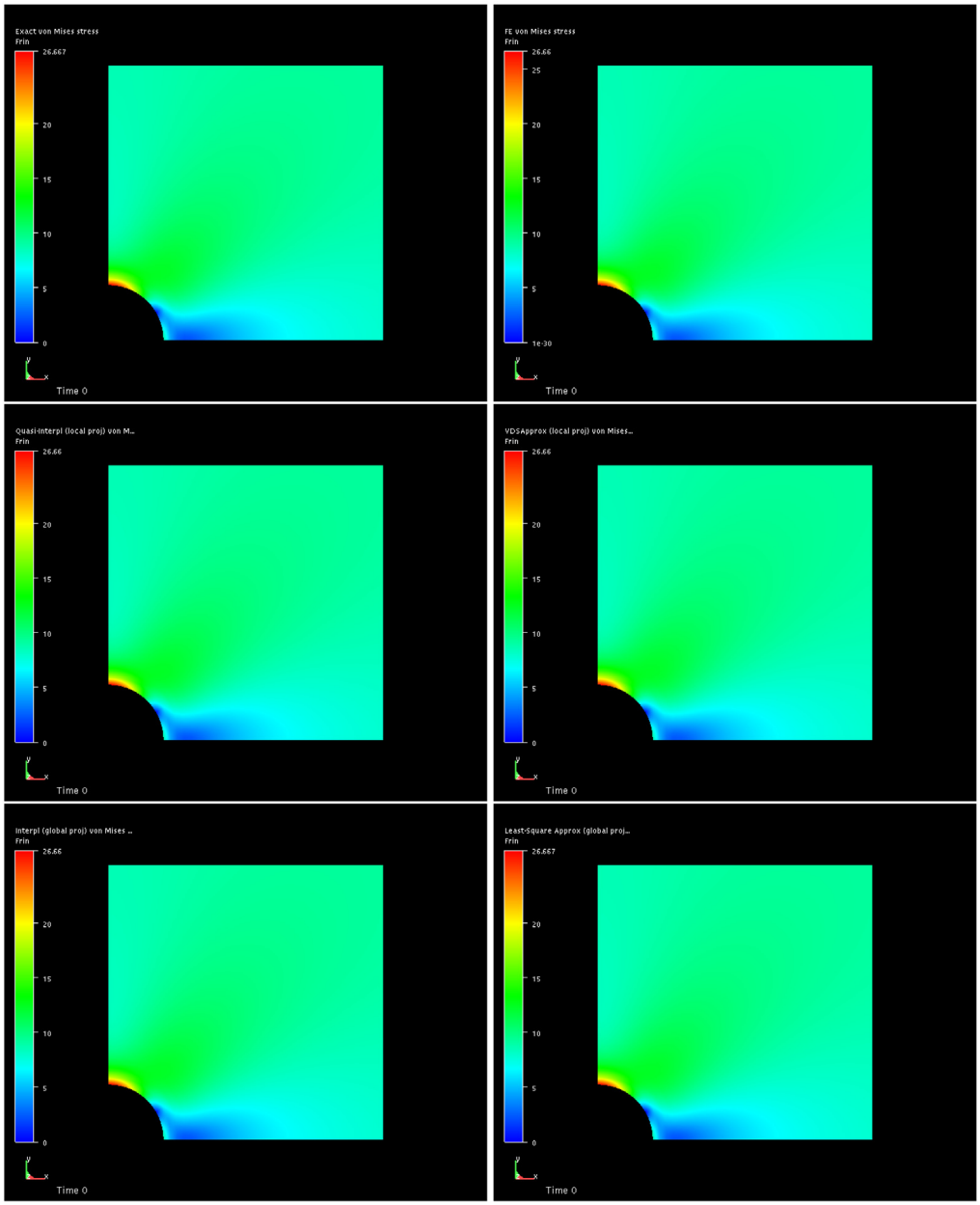

Fig. 41. 3D-Hole (C^0-line): Displaying the von Mises stresses. **Top left:** Exact solution. **Top right:** Direct finite element solution. **Middle left:** Projection method exploiting quasi-interpolation. **Middle right:** VDSA projection. **Bottom left:** Projection method exploiting spline-interpolation at Greville points. **Bottom right:** Projection method exploiting least-squares approximation.

A.1. Basic definitions and notation

A.1.1. Bernstein basis and Bézier curve

A set of Bernstein polynomial basis functions [43] of degree p is defined as $B(\xi) = \{B_{i,p}(\xi)\}_{i=1}^{p+1}$, where the Bernstein polynomials $B_{i,p}(\xi)$ for knots $\xi \in [0, 1]$ can be defined recursively as

$$B_{i,p}(\xi) = \begin{cases} 1 & i = 1,\ p = 0, \\ (1-\xi)B_{i,p-1}(\xi) + \xi B_{i-1,p-1}(\xi) & 1 < i < p+1, \\ 0 & i < 1 \parallel i > p+1. \end{cases} \tag{55}$$

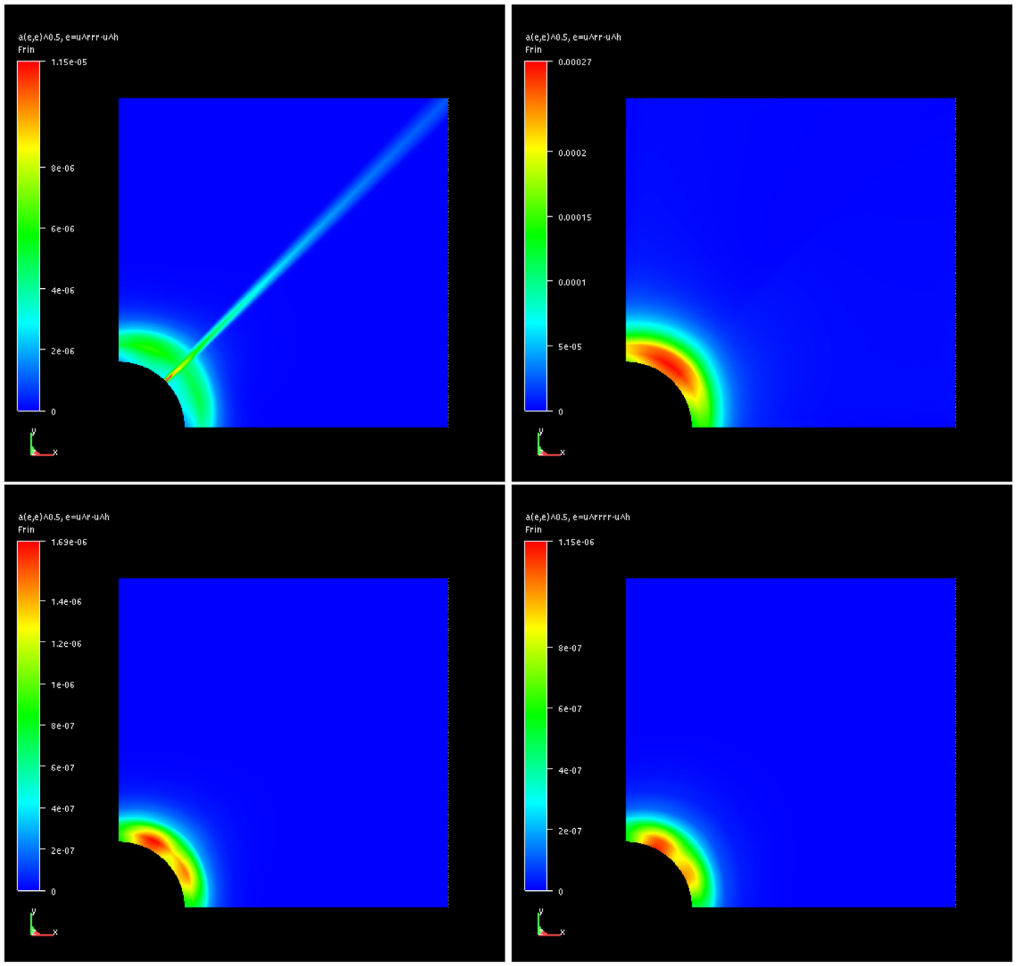

Fig. 42. 3D-Hole (C^0-line): Relative difference in von Mises stresses measured in energy norm. **Top left:** Projection: Quasi-interpolation. **Top right:** VDSA projection. **Bottom left:** Projection method exploiting spline-interpolation at Greville points. **Bottom right:** Projection method exploiting least-squares approximation.

A p-degree Bézier curve F is a linear combination of $p+1$ linear independent Bernstein polynomial basis functions with knots $\xi \in [0, 1]$ as follows

$$F(\xi) = \sum_{i=1}^{p+1} B_{i,p}(\xi) P_i \quad \xi \in [0, 1], \tag{56}$$

where $P_i \in \mathbb{R}^d$, with d as the spatial dimension, corresponds to a component of the set of vector-valued control points $P = \{P_i\}_{i=1}^{p+1}$. Since the Bernstein polynomials are non-negative on $[0, 1]$ the Bézier representation $F(\xi)$ is a convex combination of the P_i, so that $F(\xi)$ lies in the convex hull of its control points. Using this convex hull property for each Bézier element on $\xi \in [0, 1]$ a bounding box F_b for this element can be defined (using the component wise minimum and maximum of the d-dimensional control points) as

$$F_b = [\min_i P_i, \max_i P_i]_{i=1}^{p+1}. \tag{57}$$

For the later development of the Bézier extraction formulation we provide here also the curve definition (56) in matrix notation

$$F(\xi) = P^\top B(\xi) \quad \xi \in [0, 1], \tag{58}$$

where $B(\xi)$ is of dimension $d \times (p+1)$ and the matrix P has dimension $(p+1) \times d$.

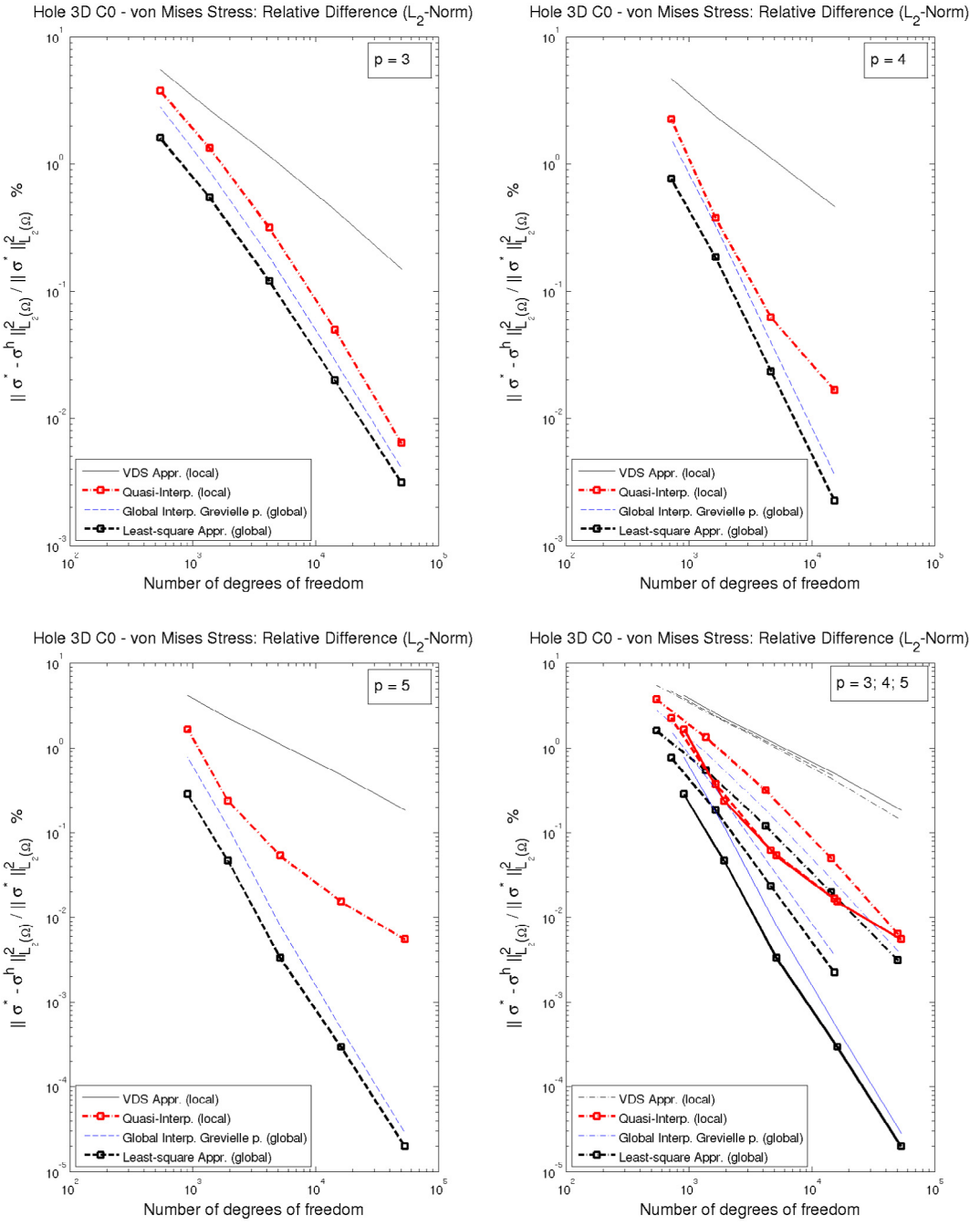

Fig. 43. 3D-Hole (C^0-line): Relative difference of von Mises stresses measured in L_2-norm: $\|\sigma^* - \sigma^h\|^2_{L_2(\Omega)} / \|\sigma^*\|^2_{L_2(\Omega)}$.

A.1.2. B-spline basis functions and B-spline curve

We denote by $N(\xi) = \{N_{i,p}(\xi)\}_{i=1}^{n}$ the set of B-spline basis functions, where the n B-spline basis functions of degree p are recursively defined by the Cox–de Boor formula [12]. For $i = 1, \ldots, n$ and degree $p = 0$ the B-spline basis functions are defined as piecewise constant:

$$N_{i,0}(\xi) = \begin{cases} 1 & \text{if } \xi_i \leq \xi < \xi_{i+1}, \\ 0 & \text{otherwise.} \end{cases} \tag{59}$$

And for $p = 1, 2, \ldots$ they are recursively defined by

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi). \tag{60}$$

Hence, a B-spline curve $F(\xi)$ is a linear combination of n linear independent piecewise polynomial B-spline basis functions $N_{i,p}$, so that

$$F(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi) P_i, \tag{61}$$

with its corresponding vector-valued control points P_i. We can rewrite Eq. (61) in matrix notation as

$$F(\xi) = P^\top N(\xi). \tag{62}$$

The curve is defined by its respective knot vector Ξ consisting of $n+p+1$ real, non-decreasing parametric coordinates $(\xi_i)_{i=1}^{n+p+1}$ on the interval $\xi \in [\xi_1, \xi_{n+p+1}]$. The support of active basis functions within a knot interval (element) $[\xi_i, \xi_{i+1}]$ depends on the chosen degree p.

A.1.3. Derivatives of B-spline basis functions

As we are concerned with the visualization of stresses as derivatives of displacement fields it is useful to understand the geometric interpretation of derivatives, therefore we want to state the formula for derivatives of B-spline basis functions as follows: For a given polynomial of degree p and knot vector $(\xi_i)_{i=1}^{n+p+1}$, the kth-derivative of the ith basis function is given by

$$\frac{d^k}{d^k\xi} N_{i,p}(\xi) = \frac{p}{\xi_{i+p} - \xi_i} \left(\frac{d^{k-1}}{d^{k-1}\xi} N_{i,p-1}(\xi) \right) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}} \left(\frac{d^{k-1}}{d^{k-1}\xi} N_{i+1,p-1}(\xi) \right). \tag{63}$$

As an example the cubic B-spline basis functions on an open knot vector and its one degree lower quadratic derivatives are shown in Fig. 66.

The first derivative of a spline curve is then given by

$$\begin{aligned} f'(\xi) &= \sum_{i=1}^{n} N'_{i,p}(\xi) c_i \\ &= \sum_{i=1}^{n} \left(\frac{p}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi) \right) c_i. \end{aligned} \tag{64}$$

A.1.4. Non-uniform rational B-splines

A NURBS curve in $\mathbb{R}^d$ can be represented by the projective transformation of a B-spline curve in $\mathbb{R}^{d+1}$ [17] using the weighting function

$$W(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi) w_i, \tag{65}$$

where $N_{i,p}(\xi)$ are the respective B-spline basis functions and $0 < w_i \leq 1$ is the ith weight.

As a consequence a NURBS curve is obtained by dividing every point of the B-spline curve by its weight. The rational B-spline basis $R_{i,p}(\xi)$ and the NURBS curve $F(\xi)$ are then defined as follows

$$R_{i,p}(\xi) = \frac{N_{i,p}(\xi) w_i}{\sum_{i=1}^{n} N_{i,p}(\xi) w_i.} = \frac{N_{i,p}(\xi) w_i}{W(\xi)}, \tag{66}$$

$$F(\xi) = \sum_{i=1}^{n} R_{i,p}(\xi) P_i = \left(\sum_{i=1}^{n} N_{i,p}(\xi) w_i P_i \right) \frac{1}{W(\xi)}, \tag{67}$$

where P_i are the control polygon coefficients for the rational B-spline curve. If we define the control point set in the projective space as $\tilde{P} = \{w_i P_i, w_i\}_{i=1}^{d+1} \in \mathbb{R}^{(d+1)}$ the B-spline curve in $\mathbb{R}^{d+1}$ that corresponds to the NURBS curve (67) in $\mathbb{R}^d$ is given by

$$F(\xi) = \tilde{P}^\top N(\xi). \tag{68}$$

A.1.5. Knot insertion

The basic underlying technique of Bézier decomposition is knot insertion [44]. Inserting one knot $\hat{\xi} \in [\xi_j, \xi_{j+1}[\in [\xi_{p+1}, \xi_{p+n-1}]$ into the existing knot vector $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{p+n-1}\}$ of a curve without changing the geometry requires the computation of new basis functions using the Cox–de Boors recursion formula as stated in (59) and (60) as well as the computation of $n+1$ new control points. Each control point – except for the first and last one – is a linear combination of the original control values as follows

$$\hat{P}_j = \begin{cases} P_j & j \leq 1 \\ P_{j-1} \cdot (1-\alpha_j) + P_j \cdot \alpha_j & j = 2, \ldots, n-1 \\ P_{j-1} & j \geq n, \end{cases} \tag{69}$$

with

$$\alpha_j = \frac{\hat{\alpha} - \alpha_j}{\alpha_{j+n} - a_j}. \tag{70}$$

We briefly indicate the key-modifications that are introduced when knots are inserted. Each time a knot is inserted the continuity of the basis function is reduced by one at the knot location (element boundary). New control variables need to be computed using (69) and (70). The new set of basis functions belongs to C^{p-m}, where m is the number of repeated knots. This set of basis functions still represents the same curve of higher regularity. Thus, knot insertion does not change the geometric properties of the curve.

A.2. Spline interpolation and approximation methods

Spline notations used in this section are introduced in Appendix A.1.

We exploit approximation and interpolation methods to generate higher-order spline functions that match spline derivative functions representing stress quantities in form of scalar field values.

Interpolation refers to the problem to construct a B-spline such that the curve passes through a set of distinct given points, where parameter values, knots and the polynomial degree are given, while approximation methods deal with the question how functions can be best represented by simpler functions.

In the following we classify the techniques into local and global approximation methods. Global methods like spline interpolation and least-squares approximation, are methods where we have to solve a global linear system of equations that leads to the B-spline coefficients for given data. For these methods the B-spline coefficients depend on all initial data points. Local methods like cubic Hermite and the variation diminishing spline approximation refer to approaches where the value of the coefficients is directly given in terms of known values of the function which has to be interpolated. Here, the B-spline coefficients depend only on neighboring points of the support of the corresponding B-spline [13]. Another local approach is the "Superconvergent Patch Recovery" technique initially introduced by Zienkiewicz and Zhu for a posteriori error estimation for the classical finite element method. Similar techniques have been developed recently for splines by Kumar, Kvamsdal and Johannessen [32].

The choice of the parameter values ξ and the knot vector Ξ affect the parametrization and therefore the shape of a spline curve $f(\xi)$. Instead of equally spaced knots it is recommended to determine the knots as a result of the parameter distribution.

Greville points. The concept of Greville points plays a key role in the construction of the proposed projection methods. Greville points lie at the mean location of p consecutive knots in the knot vector for each spline basis function of degree p and are defined as

$$\xi_i^* = \frac{1}{p} \sum_{k=i+1}^{i+p} \xi_k, \quad i = 1, 2, \ldots, n. \tag{71}$$

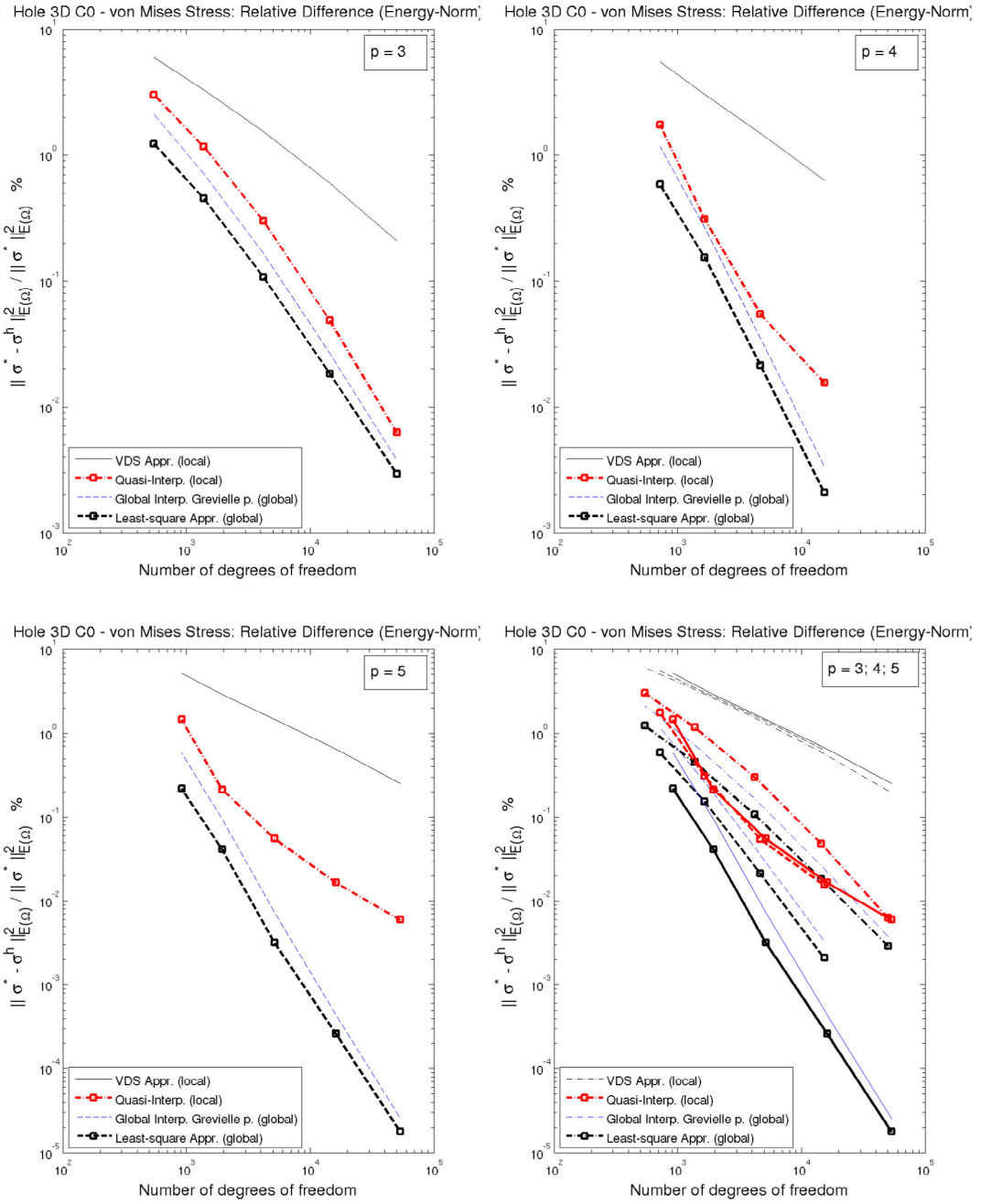

Fig. 44. 3D-Hole (C^0-line): Relative difference of von Mises stresses measured in energy norm: $\|\sigma^* - \sigma^h\|^2_{E(\Omega)} / \|\sigma^*\|^2_{E(\Omega)}$.

Note that the above expression is not defined for $p = 0$ and that the first and last knots in the knot vector are excluded, though the fact that starting from degree $p = 1$ the multiplicity of the open knot vectors has to be taken into account.

Using Greville points it is guaranteed that every knot span contains at least one ξ_i and the solution of a system which consists of a set of linear equations (61) becomes positive definite with a semi-bandwidth less than p and non-negative. Greville points are therefore often used in B-spline collocation applications and are also known as Marsden–Schoenberg points.

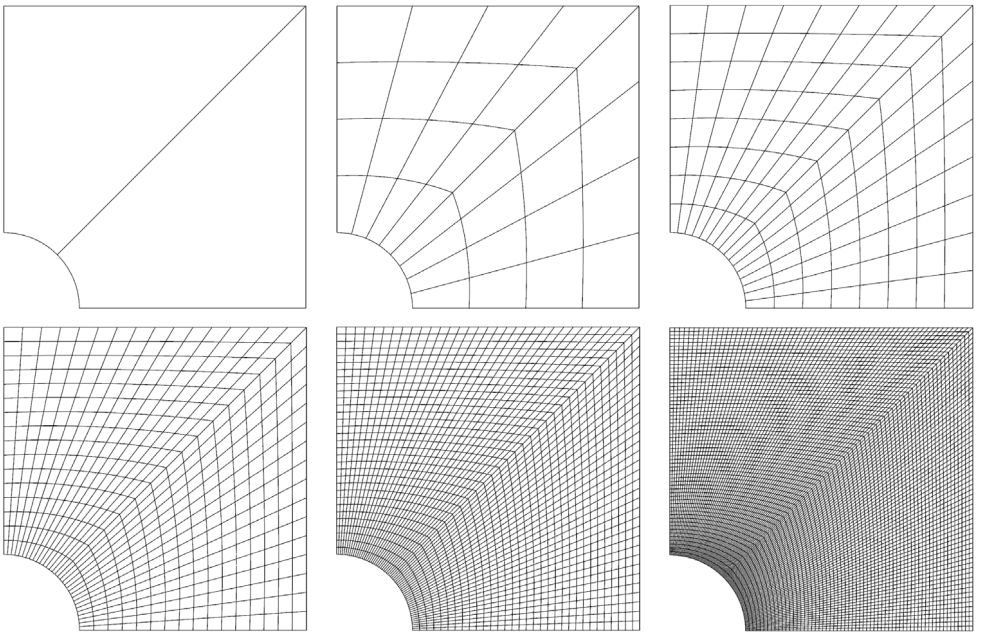

Fig. 45. 3D-Hole (2 patches): Displaying uniform mesh refinements. From 0, 3, 7, 15, 31 and 61 interior knots per element.

A.2.1. Global methods

We investigate in the following the common spline interpolation as well as the least-squares spline approximation method.

Spline interpolation. We seek a spline function f that matches given data values y_j at given points x_j, such that

$$f(x_j) = y_j, \quad j = 1, \dots, m, \tag{72}$$

expressed in terms of m basis functions

$$f(x_j) = \sum_{i=1}^{m} N_{i,p}(x_j) c_i = y_j. \tag{73}$$

Eq. (73) can be reformulated as a set of linear systems of equations

$$f(x) = Ac = y \tag{74}$$

with m unknowns and m equations

$$f(x) = Ac = \begin{pmatrix} N_{1,p}(x_1) & \cdots & N_{m,p}(x_1) \\ \vdots & \ddots & \vdots \\ N_{1,p}(x_m) & \cdots & N_{m,p}(x_m) \end{pmatrix} \begin{pmatrix} c_1 \\ \vdots \\ c_m \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} = y. \tag{75}$$

To ensure an existing and unique solution of the system of equations we utilize the *Schoenberg–Whitney Theorem* that states the following necessary and sufficient conditions:

Assume that $x := (x_1 < \cdots < x_m)$ is a strictly increasing sequence and let $\Xi = (\xi_i)_{i=1}^{n+p+1}$ be an open knot vector. Then the collocation matrix

$$A := (N_{j,p}(x_i) : i, j = 1, 2, \dots, m)$$

is invertible if and only if all its diagonal entries are nonzero, i.e. if and only if the following nesting conditions hold:

$$\xi_i < x_i < \xi_{i+p+1}, \quad i = 1, 2, \dots, m,$$

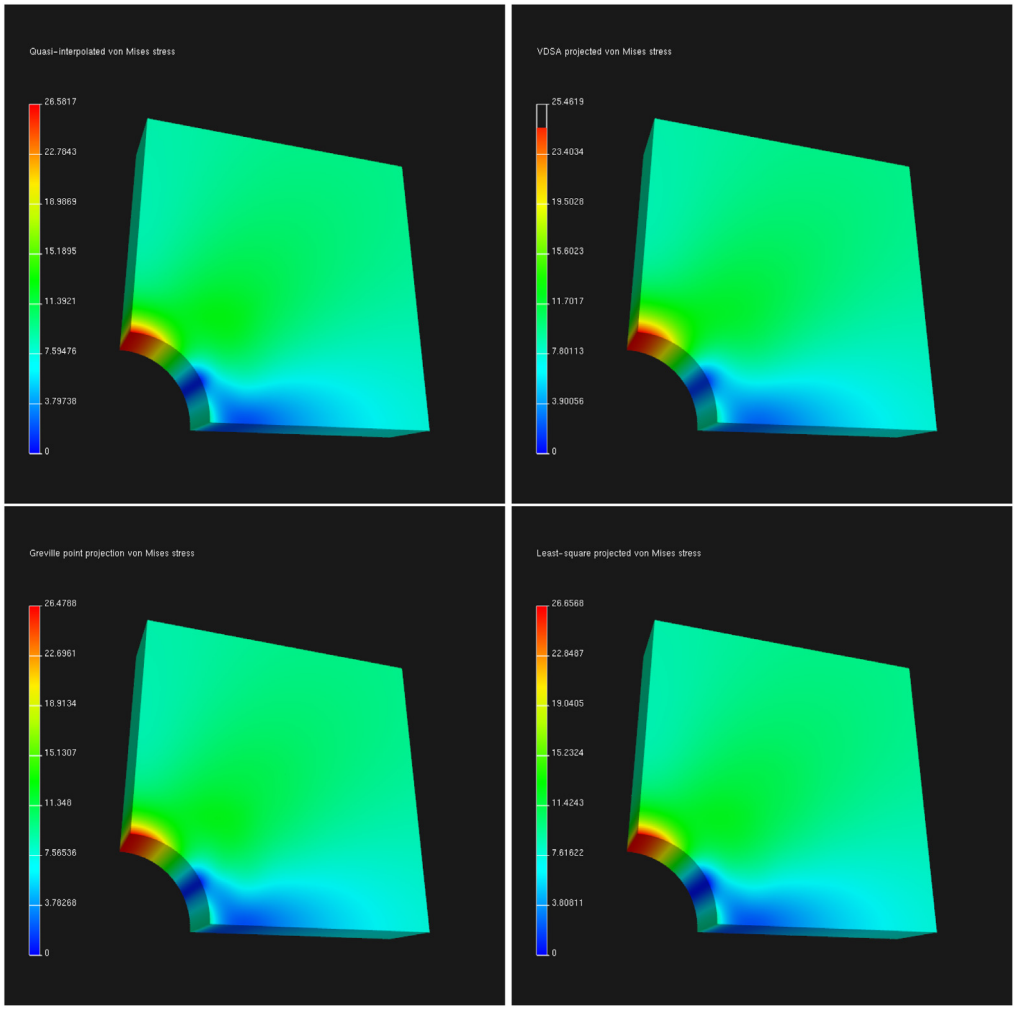

Fig. 46. 3D-Hole (2 patches): Displaying von Mises stresses. **Top left:** Projection: Quasi-interpolation. **Top right:** VDSA projection method. **Bottom left:** Projection method exploiting spline-interpolation at Greville points. **Bottom right:** Projection method exploiting least-squares approximation.

we allow $x_i = \xi_i$ if $\xi_i = \cdots = \xi_{i+p}$. If knot sequence and degree are already given, then the sequence of Greville points ξ^* is a good choice as data point sequence x and it certainly satisfies the Schoenberg–Whitney conditions.

Least-squares spline approximations result in curves such that the error at the data points is minimized instead of passing exactly through the points as this is the case for interpolation methods. The least-squares approach defines u by minimizing the L_2-norm

$$\min_{\tilde{u}\in S} \|u(x) - u^h(x)\|_{L_2} \tag{76}$$

this results in the minimization of the following integral

$$\min \frac{1}{2} \int_\Omega (u(x) - u^h(x))^2 dx. \tag{77}$$

By using the common test function approximation

$$u^h = N\tilde{u} \tag{78}$$

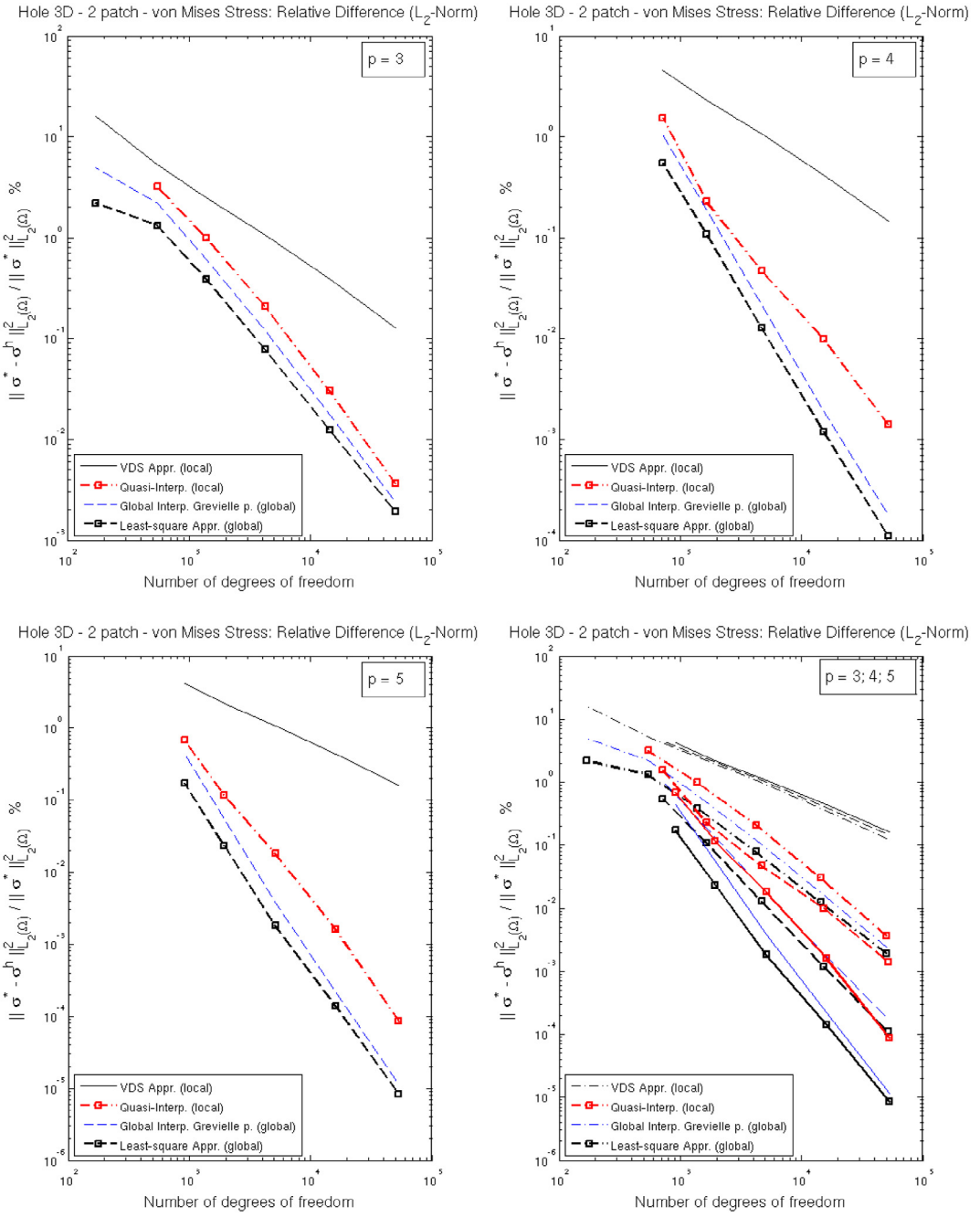

Fig. 47. 3D-Hole (2 patches): Relative differences of von Mises stresses measured in L_2-norm: $\|\sigma^* - \sigma^h\|^2_{L_2(\Omega)} / \|\sigma^*\|^2_{L_2(\Omega)}$.

we can substitute and derivate

$$\frac{1}{2}\int_\Omega (u - N\tilde{u})^2 dx = \int_\Omega N^\top (u - N\tilde{u}) dx = 0 \tag{79}$$

$$\int_\Omega N^\top N\tilde{u} dx \quad = \int_\Omega N^\top u dx. \tag{80}$$

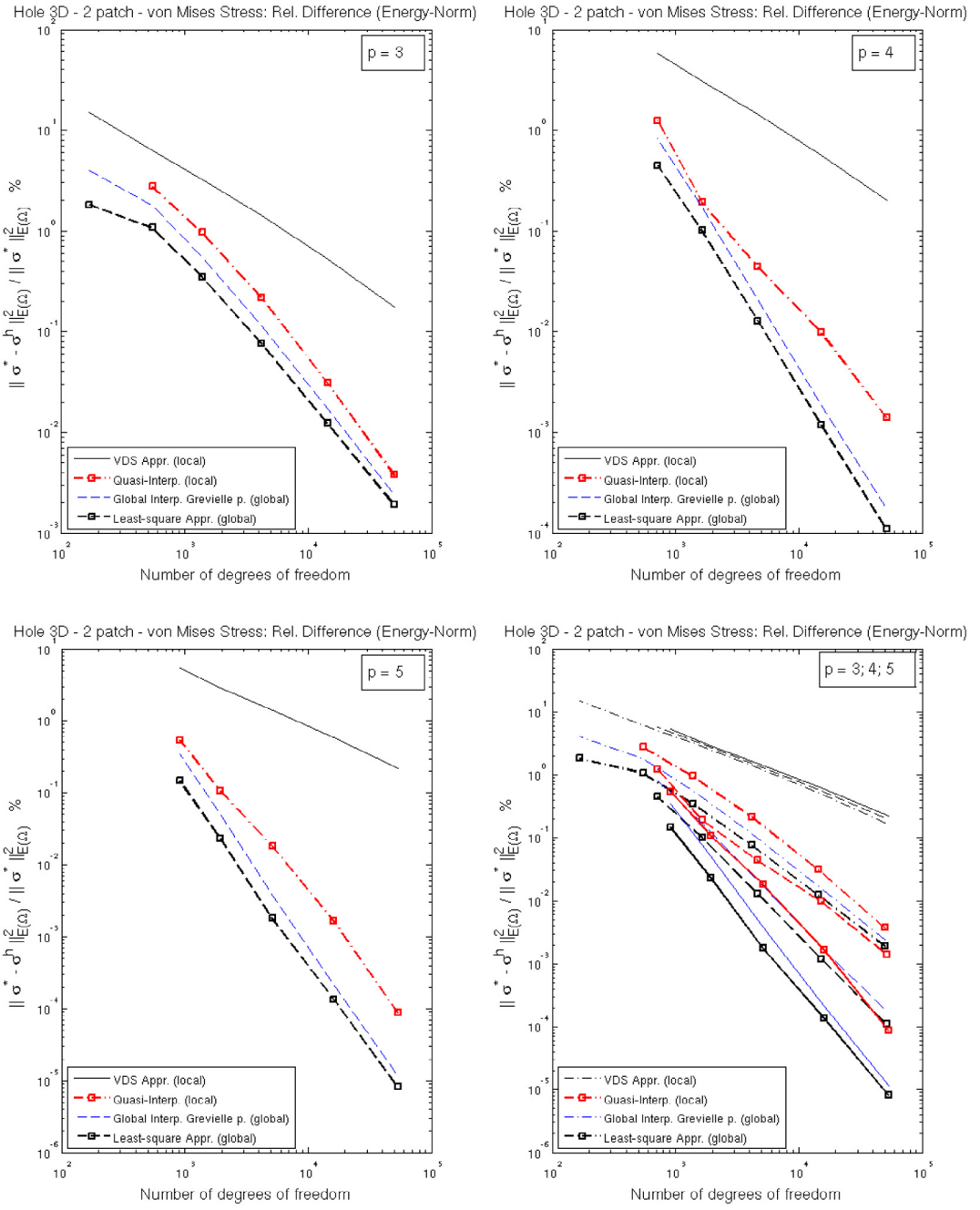

Fig. 48. 3D-Hole (2 patches): Relative differences of von Mises stresses measured in energy norm: $\|\sigma^* - \sigma^h\|^2_{E(\Omega)} / \|\sigma^*\|^2_{E(\Omega)}$.

By using the Gauss quadrature rule to approximate the integral this leads to the solution of a linear system

$$N^\top N \tilde{u} = N^\top u \tag{81}$$

in n unknowns $\tilde{u}$ where the function u and the n basis functions

$$(N^\top N)_{i,j} = \int_\Omega \varphi_i(x)\varphi_j(x)dx \tag{82}$$

are evaluated at the respective Gauss points.

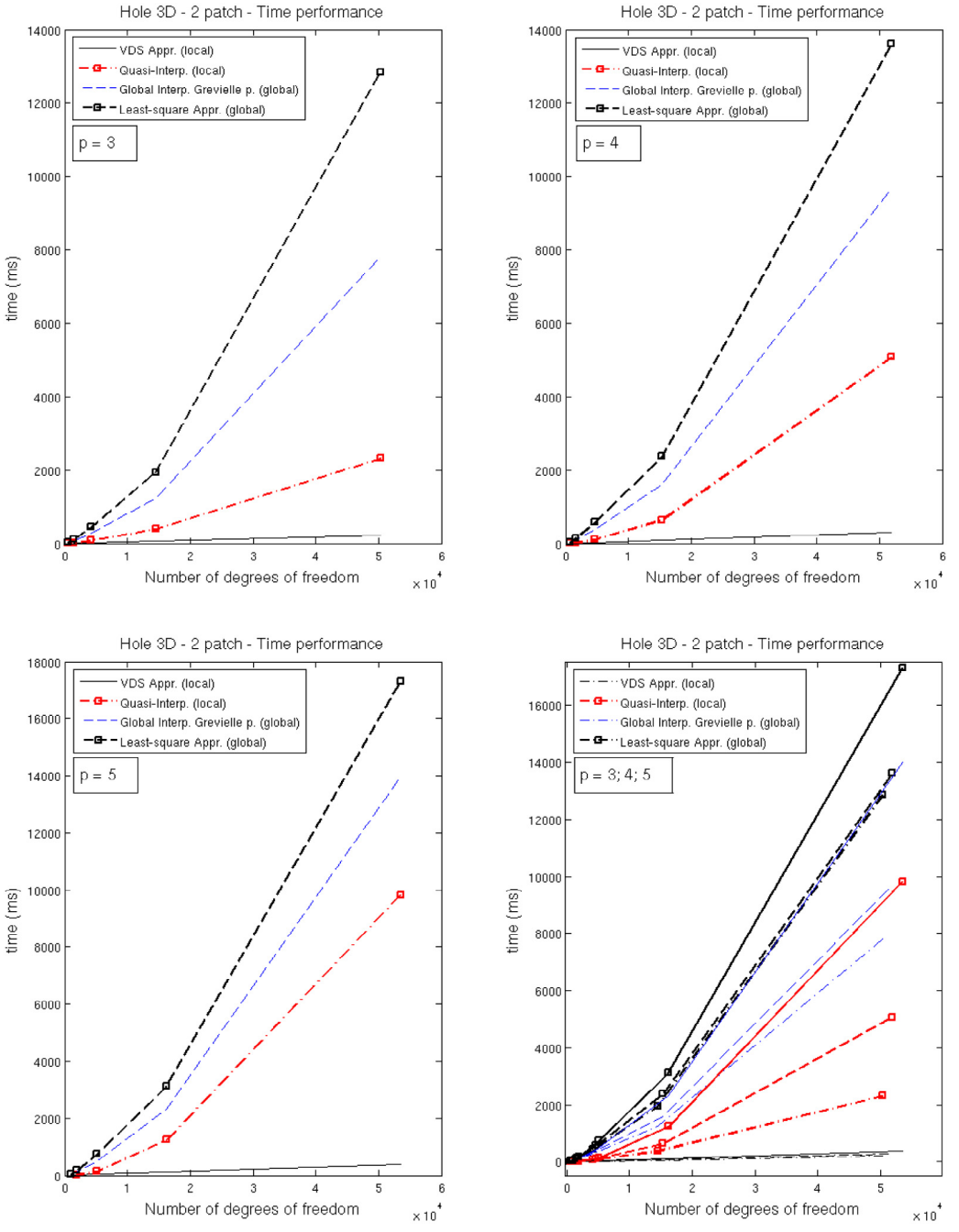

Fig. 49. 3D-Hole (2 patches): Cost comparison. CPU time performance for the different projection techniques.

The coefficient matrix $M = N^\top N$ is symmetric and positive semi-definite. The matrix is positive definite and has therefore a unique solution if and only if N has linear independent columns.

A.2.2. Local methods

Variation Diminishing Spline Approximation (VDSA). A useful method to obtain a local spline approximation to a function f defined on an interval $[a, b]$ with shape preserving property is given as follows. Let f be a given continuous function on the interval $[a, b]$, let p be the degree of the spline basis functions and let $\Xi = (\xi_i)_{i=1}^{n+p+1}$ be

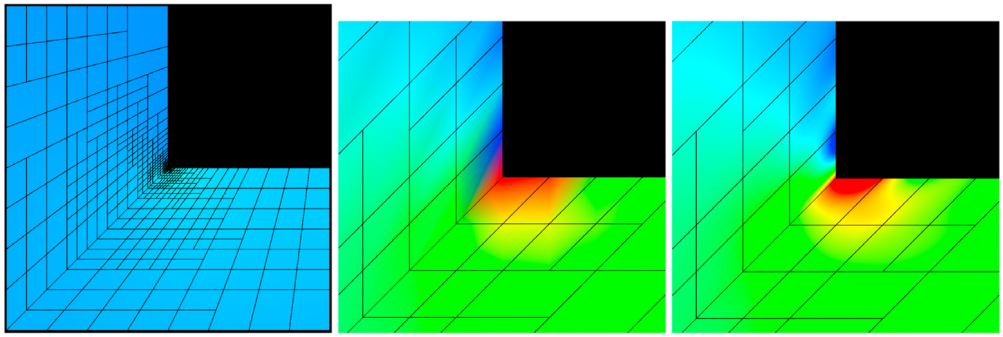

Fig. 50. 2D LR B-splines adaptive refinement: Computed von Mises stresses shown on the mesh obtained at refinement level 9. **Left:** Entire Lshape domain. **Center:** Zoom into the region of interest at the singularity point of the Lshape. A linear finite element mesh is used for tessellation. **Right:** Zoom into the region of interest at the singularity point of the Lshape exploiting pixel-accurate spline evaluation.

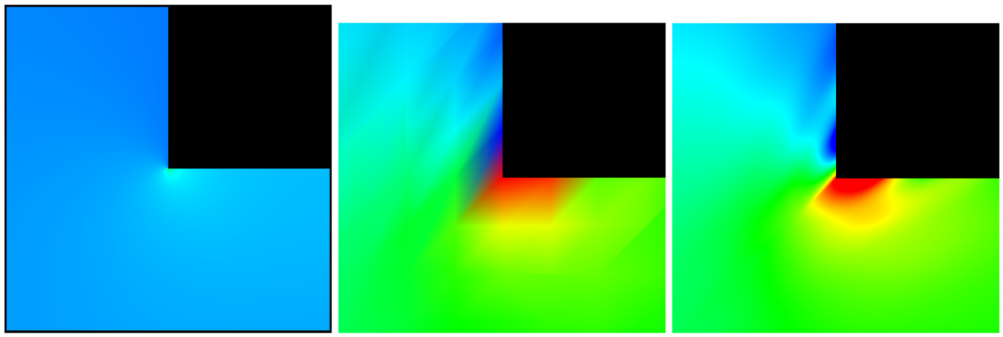

Fig. 51. 2D LR B-splines adaptive refinement: Computed von Mises stresses obtained at refinement level 9. **Left:** The entire Lshape domain. **Center:** Zoom into the region of interest at the singularity point of the Lshape. A linear finite element mesh is used for tessellation. **Right:** Zoom into the region of interest at the singularity point of the Lshape using pixel-accurate spline evaluation.

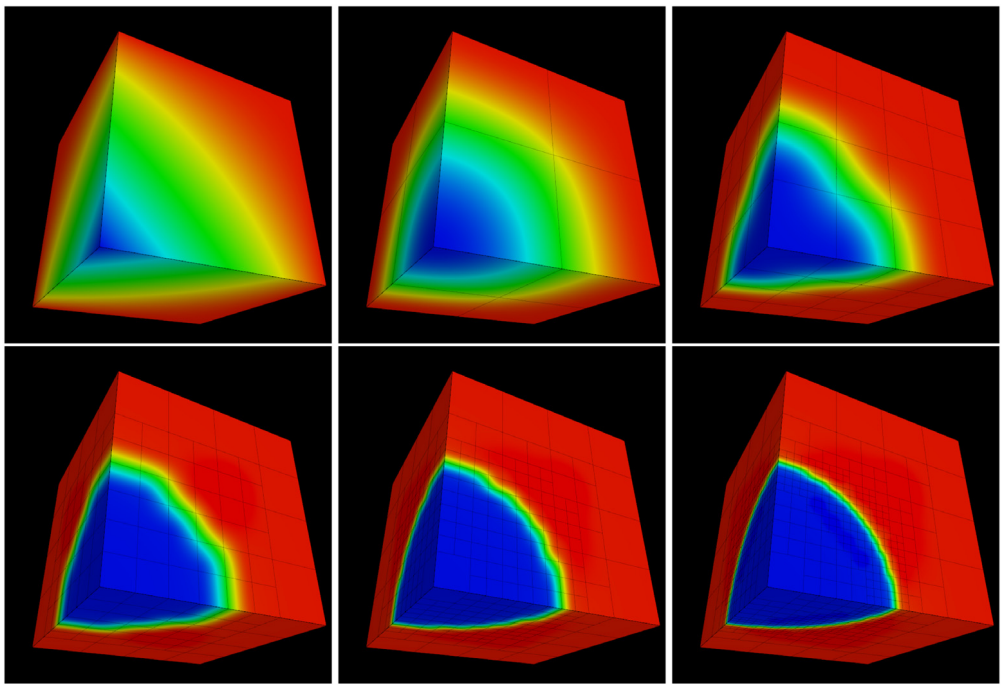

Fig. 52. 3D Cube: Adaptive local refinement. Pixel-accurate rendering results are shown for 5 refinement levels.

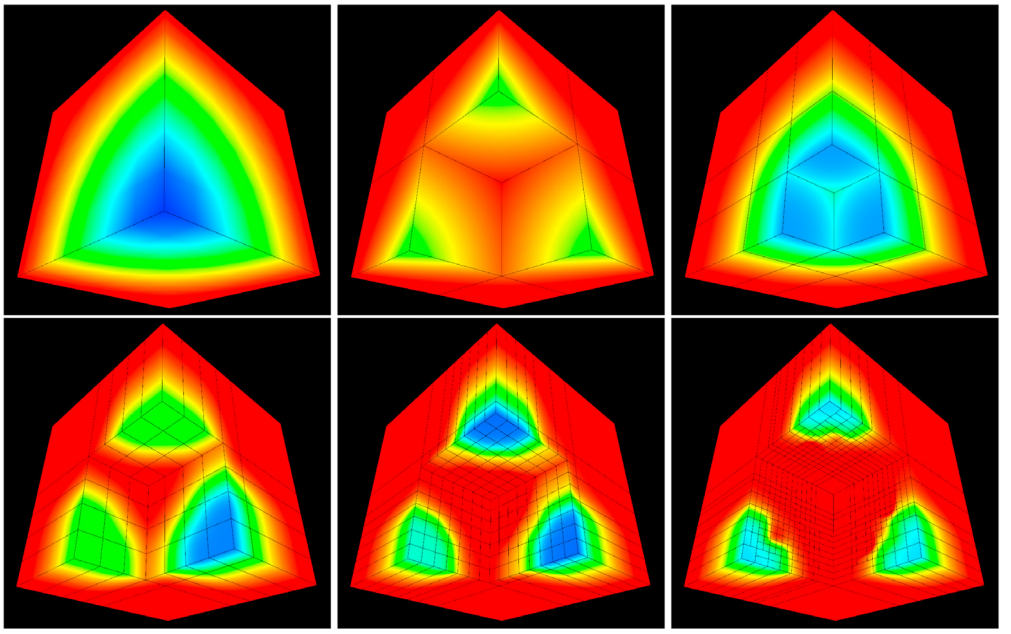

Fig. 53. 3D Cube: Adaptive local refinement. Pixel-accurate rendering results are shown for 5 refinement levels allowing insights into the 3D structure.

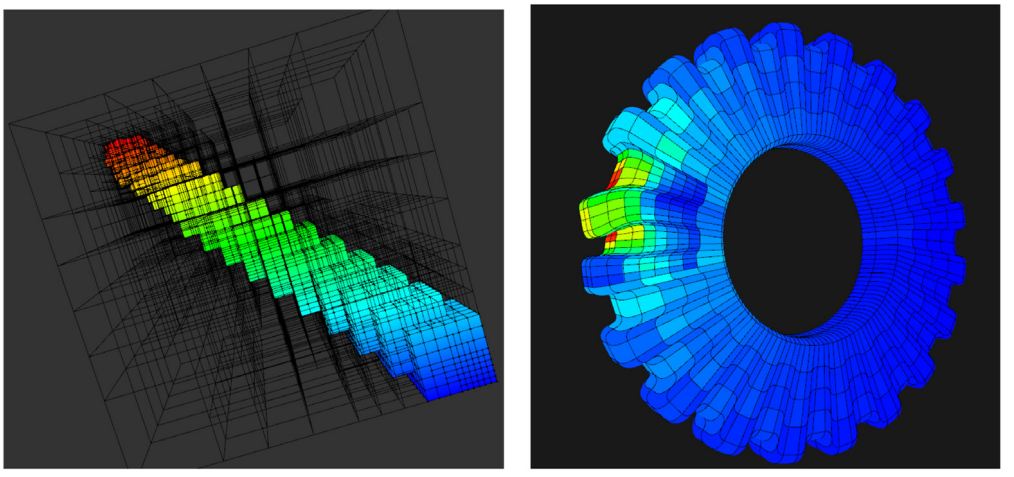

Fig. 54. **Left:** Pixel-accurate rendering. 3D volume-element-view through a LR B-spline cube. **Right:** 3D LR B-splines: Volume element error plot.

a $p+1$-regular knot vector with boundary knots $\xi_{p+1} = a$ and $\xi_{n+1} = b$. The spline is then given by the VDSA

$$(Vf)(x) = \sum_{j=1}^{n} f(\xi_j^*) N_{j,p}(x) \tag{83}$$

of degree p of f on the knot vector Ξ, with ξ_j^*, $j = 1, \ldots, n$ the Greville abscissa.

Preservation of bounds on a function: To preserve the maximum and minimum values of a function under approximation the following is stated: Let f be a spline in some spline space $S_{p,\Xi}$ of dimension n. Then f is bounded by its smallest and largest B-spline coefficients

$$min\{c_i\} \leq \sum c_i N_i(x) \leq max\{c_i\} \quad \forall x \in [\xi_{p+1}, \xi_{n+1}). \tag{84}$$

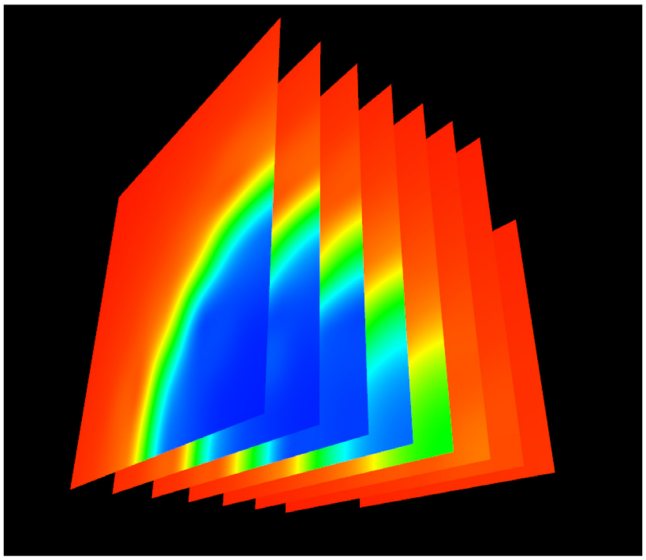

Fig. 55. Exploiting the ray casting approach we are able to render cutting planes through a LR B-spline volume.

With this statement it is guaranteed that bounds on a function are preserved by its variation diminishing approximation: Let f be a function that satisfies

$$f_{min} \leq f(x) \leq f_{max} \quad x \in \mathbb{R}. \tag{85}$$

Then the VDSA to f has the same bounds

$$f_{min} \leq (Vf)(x) \leq f_{max} \quad x \in \mathbb{R}. \tag{86}$$

Quasi-interpolation. A more flexible but more expensive family of methods is known as quasi-interpolation schemes. Using a general two step scheme any combination of known approximation and interpolation schemes can be used to generate a new local approximation method [13].

Quasi-interpolation methods are based on the local support property of splines stating that the ith spline basis N_i is non-zero only within the interval $[\xi_i, \xi_{i+p+1}]$.

From this it follows that on the interval $[\xi_j, \xi_{j+1}]$ a sub-interval of $[\xi_i, \xi_{i+p+1}]$ where $\xi_j < \xi_{j+1}$ exist with $p+1$ non-zero basis functions.

In order to compute an approximation

$$g(x) = \sum_{j=1}^{n} N_{j,p}(x) b_j \tag{87}$$

in $S_{p,\Xi}$ to a given function f the n coefficients b_j are computed locally.

Choosing a local method an approximation

$$g_I(x) = \sum_{i=j-p}^{j} N_{i,p}(x) c_i,$$

on the knot interval $I = [\xi_j, \xi_{j+1}]$, where x is restricted to the interval I, is determined. We define as new coefficient b_j one of the coefficients of g_I with $b_j = c_j$. By repeating these local step n times we obtain n coefficients for Eq. (87).

Note, that the interval I is not restricted to one knot interval, it is required that I is nonempty and is defined in a subset of $[\xi_i, \xi_{i+p+1}]$. A variety of interpolation or approximation methods can be used. Methods like spline interpolation or least-squares approximation are used within the quasi-interpolation setting, a linear system of equations has to be solved, where as using the VDSA method the coefficients are given directly, as described in the previous section.

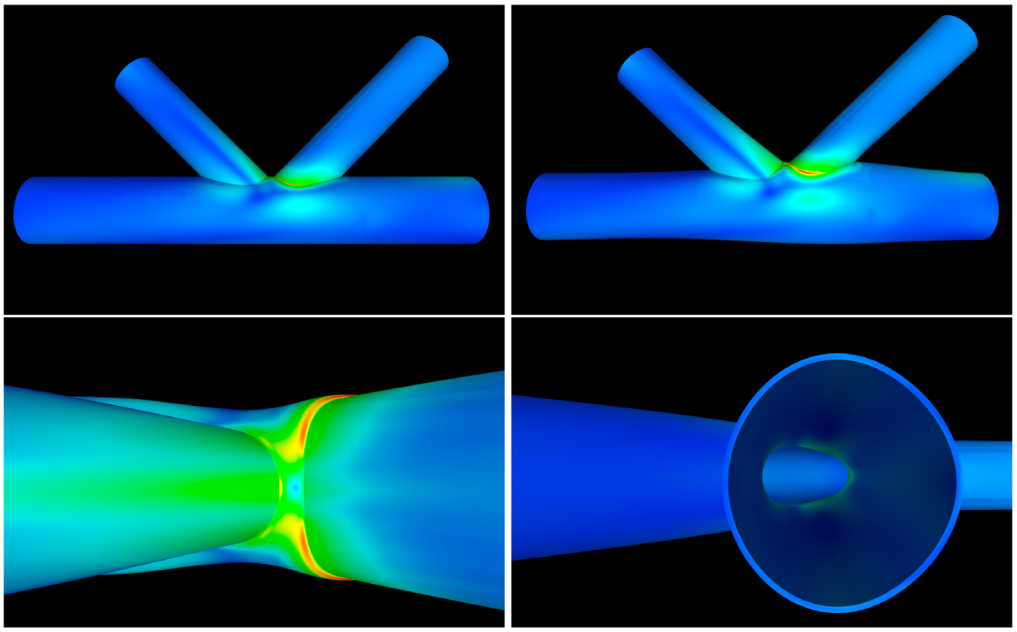

Fig. 56. K-joints: Linear elasticity result von Mises stresses.

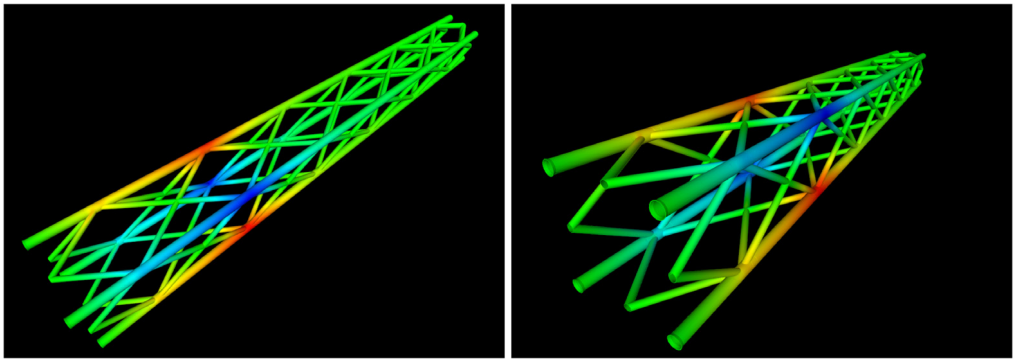

Fig. 57. Jacket: Linear elasticity result von Mises stresses.

A.2.3. Comparison

Fig. 67 shows a quadratic example function fitted by a typical spline interpolation, a least-squares method, a VDSA and a quasi-interpolation scheme, all four variants plotted with exact relative difference error

$$\text{RDE} = \frac{\|f - g\|}{\|f\|}.$$

We observe that the global approximation methods yield the best fit, where the least-squares method is superior.

A.3. Locally Refined (LR) B-splines

LR B-splines were introduced by Dokken et al. [2] and consist of smooth, piecewise polynomial basis functions that constitute a partition of unity and facilitate local h-refinement. Adaptive local refinement techniques for isogeometric finite elements based on LR B-splines was developed by Johannessen et al. [3] and has been investigated and utilized together with newly developed a posteriori error estimates, see [45,32], divergence conforming discretization for Stokes problem [46] and for porous media flow [47]. A comparison of LR B-splines towards hierarchical splines may be found in [48].

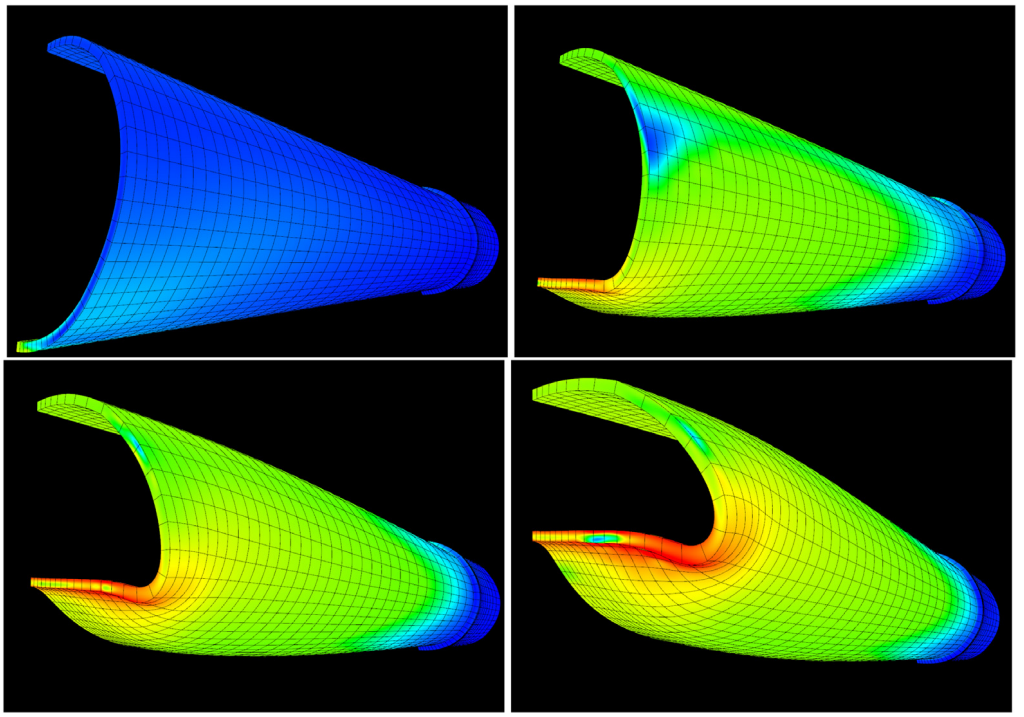

Fig. 58. Pipe contact: Contact between a subsea oil pipeline and trawl gear from a fishing vessel simulated by IFEM using a non-linear mortar based contact algorithm [35]. Pixel-accurate spline evaluation of the von Mises stresses.

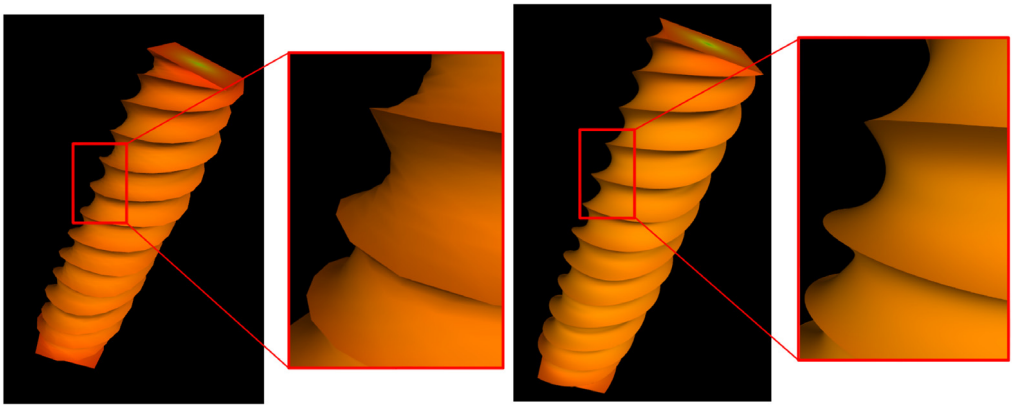

Fig. 59. Twisted bar: Non-linear torsional deformation of a bar with the resulting von Mises stresses. **Left:** Tessellation using linear finite elements, with 5 interior visualization knots per element. **Right:** Pixel-accurate spline evaluation.

In order to be self-contained we briefly explain the mathematical details of LR B-splines below.

A.3.1. Defining a tensor product B-spline mesh as LR-mesh

In this section we use a 2D example in order to illustrate the transformation of a tensor product mesh into a locally refined mesh. For a common 2D tensor product B-spline mesh the global knot vectors for each parameter direction are defined as $\Xi = [\xi_1, \xi_2, \ldots, \xi_{p+n+1}]$ and $\mathcal{H} = [\eta_1, \eta_2, \ldots, \eta_{q+m+1}]$, with degree p, q, respectively so that $n \times m$ basis functions are defined over the tensor product B-spline mesh.

An example mesh is shown in Fig. 68 with global knot vectors Ξ and $\mathcal{H}$. They are defined as $\Xi = [0, 0, 0, 2, 2, 2]$ and $\mathcal{H} = [0, 0, 0, 1, 2, 2, 2]$ with degree $p = q = 2$ respectively. Note, that this mesh consists of two elements

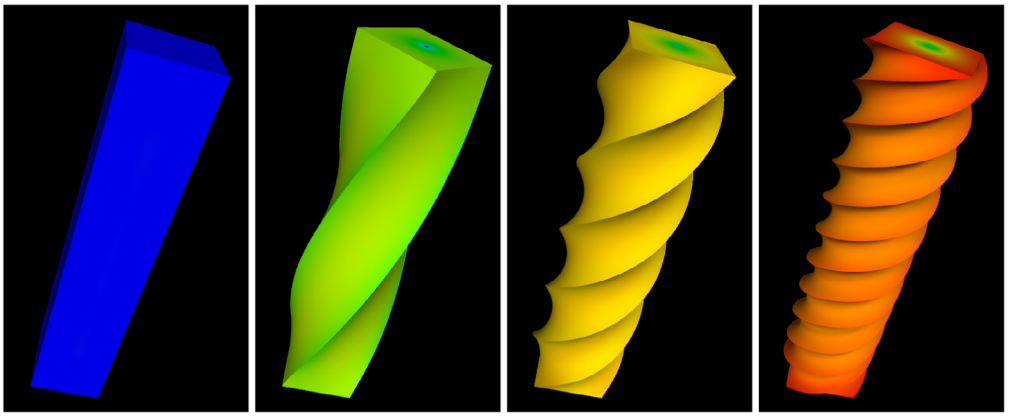

Fig. 60. Twisted bar: Non-linear torsional deformation of a bar with the resulting von Mises stresses.

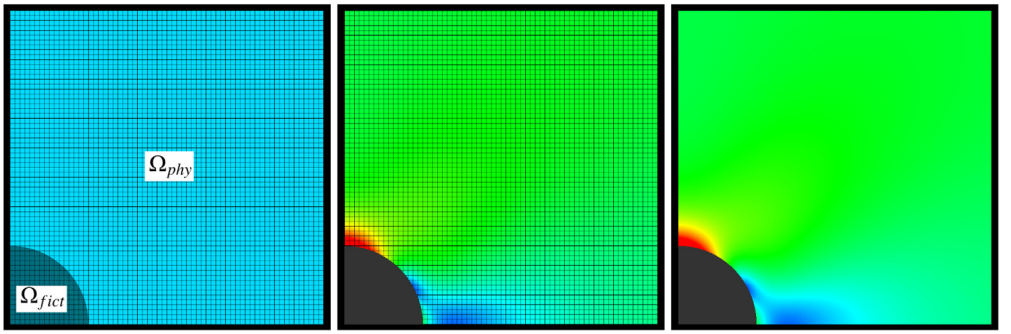

Fig. 61. 2D-Hole (Immersed boundary method): Circular hole example. **Left:** Defining the domain $\Omega = \Omega_{phy} + \Omega_{fict}$ in physical and fictitious domain. **Center:** After the trimming operation in the fragment shader. The analysis result is shown on the physical domain with the Cartesian grid structure. **Right:** Solution of the immersed boundary problem: Von Mises stress representation.

forming one patch. The $n \times m = 3 \times 4$ basis functions which have support over these elements are shown symbolically as yellow dots in the left image of Fig. 68. The basis functions are multiplied by corresponding control points and summed to construct a geometrical object – a 2D surface – depicted in the center image of Fig. 68 (control points are marked as black dots). The visualization of a classical Poisson problem computation for this geometry is visualized in the most right image of Fig. 68.

In order to be able to convert a general tensor product mesh into a LR-mesh certain definitions are noted:

A.3.2. Defining the local knot vectors

The minimal support of a bivariate B-spline basis function is defined over $(p+2) \times (q+2)$ knots using two local knot vectors Ξ_i and $\mathcal{H}_i$ with $i = 1, 2, \ldots, N$. We define a local knot vector as a non-decreasing knot sequence

$$\Xi_i = [\xi_i, \xi_{i+1}, \ldots, \xi_{p+i}, \xi_{p+i+1}].$$

For a tensor product mesh n unique such local knot vectors for the first parameter direction and m for the second parameter direction can be defined.

For the 2D example shown in Fig. 68 the unique local knot vectors in the first parameter direction are [0, 0, 0, 2], [0, 0, 2, 2] and [0, 2, 2, 2] and in the second parameter direction are [0, 0, 0, 1], [0, 0, 1, 2], [0, 1, 2, 2] and [0, 2, 2, 2].

A.3.3. Defining the local parameter domain

Each set of local knot vectors $\{\Xi_i, \mathcal{H}_i\}_{i=1}^{N}$ define a two-dimensional local basis function domain

$$\Omega_i = (\Xi_i \cdot \mathcal{H}_i).$$

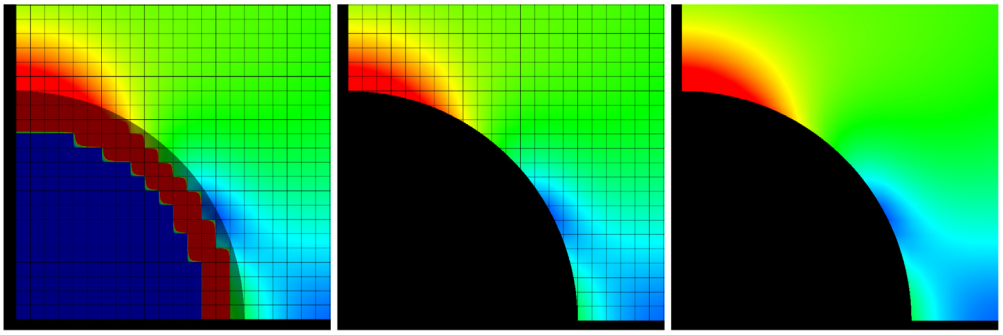

Fig. 62. 2D-Hole (Immersed boundary method): Circular hole example. **Left:** Zooming in the interesting domain situation. We see here the analysis result on the physical domain and gray shaded the artifacts that occur during computation within the fictitious domain. **Center:** After the trimming operation in the fragment shader: Analysis result shown on the physical domain with the Cartesian grid structure. **Right:** Solution of the immersed boundary problem (zoom-in illustration).

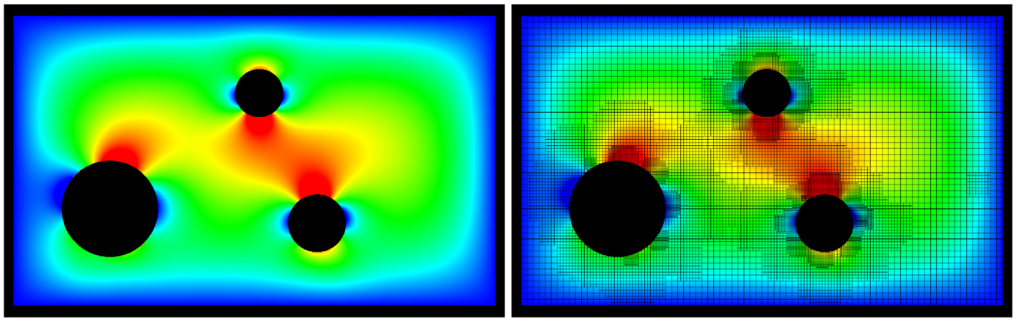

Fig. 63. ThinPlate (Immersed boundary method): Von Mises stresses obtained with Kirchhoff–Love thin plate theory using the immersed boundary method as well as adaptive refinement exploiting LR B-splines.

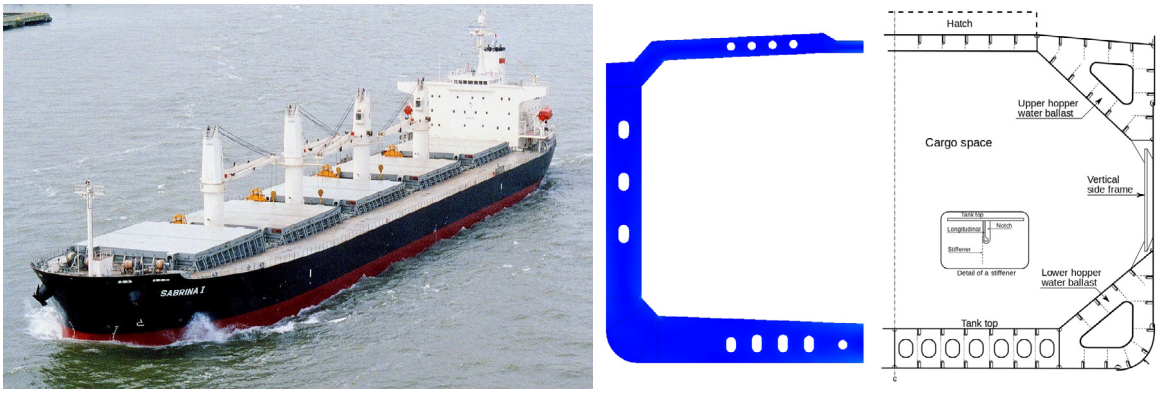

Fig. 64. MidShipSection (Immersed boundary method): Immersed boundary techniques for easier modeling. **Left:** SabrinaI: A typical modern handy-max bulk carrier with 5 holds, 5 hatches and 4 cranes (pic: with courtesy of www.wikipedia.org). **Center:** Model of the mid ship section with holes. **Right:** Construction sketch.

For the example shown in Fig. 68, 12 local parameter domains exist and they are denoted as follows

$$\begin{aligned}
[\Xi_1; \mathcal{H}_1] &= [0002][0001] \\
[\Xi_2; \mathcal{H}_2] &= [0022][0001] \\
\vdots \qquad\qquad &\quad \vdots \\
[\Xi_{12}; \mathcal{H}_{12}] &= [0222][1222].
\end{aligned}$$

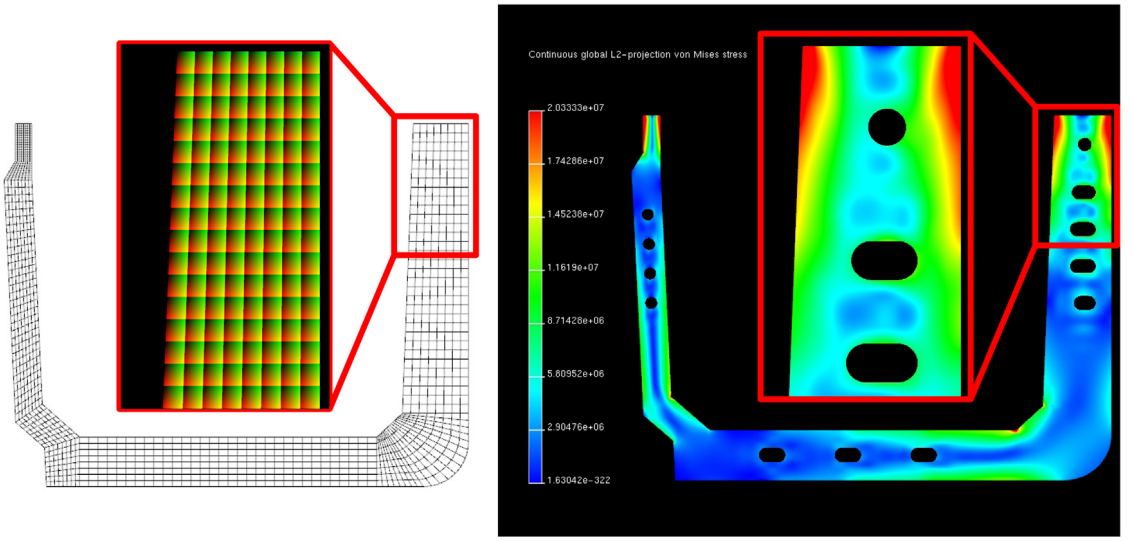

Fig. 65. MidShipSection (Immersed boundary method): Von Mises stresses obtained with use of the immersed boundary method. **Left:** Cartesian grid. The grid is illustrated as color illustration showing the fictitious domain concept. **Right:** The fictitious domain is removed using trimming operations on the fragment processor, programmed in the fragment shader. In total are five circular holes and seven ellipsoids trimmed away.

Using these local parameter domains 12 individual basis functions are defined.

A.3.4. Defining an individual LR B-spline basis function

Over each local parameter domain Ω_i a bivariate LR B-spline basis function $N_{(p,\Xi_i;q,\mathcal{H}_i)}(\xi,\eta)$ with minimal support can be defined. We derive the bivariate form by forming the tensor product of the two univariate B-spline basis functions

$$N_{(p,\Xi_i;q,\mathcal{H}_i)}(\xi,\eta) = N_{p,\Xi_i}(\xi) N_{q,\mathcal{H}_i}(\eta). \tag{88}$$

The univariate B-spline basis function $N_{p,\Xi_i}(\xi)$ is recursively defined using the Cox–de Boor formula (59)–(60) on a local knot vector Ξ_i

$$N_{p,\Xi_i}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{p-1,[\xi_i,\ldots,\xi_{i+p}]}(\xi) \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{p-1,[\xi_{i+1},\ldots,\xi_{i+p+1}]}(\xi), \tag{89}$$

where for $(p = 0)$

$$N_{0,[\xi_i,\xi_i+1]}(\xi) = \begin{cases} 1 & \xi_i \le \xi < \xi_{i+1} \\ 0 & \text{otherwise} \end{cases}. \tag{90}$$

The $N = n \times m = 3 \times 4$ LR B-spline basis functions with minimal support for our small example are shown symbolically as yellow dots in the left image of Fig. 68.

A.3.5. Defining the LR B-spline mesh

We defined the tensor product mesh as a LR-mesh by rearranging the tensor product mesh as a set of local parameter domains Ω_i offering minimal support to each individual basis function over this parameter domain.

A.3.6. The LR B-spline refinement — Knot insertion

If a knot $\hat{\xi}$ is inserted into a local knot vector Ξ between the knots ξ_k and ξ_{k+1}, where $\xi_k \le \hat{\xi} \le \xi_{k+1}$, the resulting local knot vectors are defined as

$$\Xi_1 = [\xi_1, \xi_2, \ldots, \xi_k, \hat{\xi}, \xi_{k+1}, \ldots, \xi_p, \xi_{p+1}], \tag{91}$$

$$\Xi_2 = [\xi_2, \xi_3, \ldots, \xi_k, \hat{\xi}, \xi_{k+1}, \ldots, \xi_{p+1}, \xi_{p+2}]. \tag{92}$$

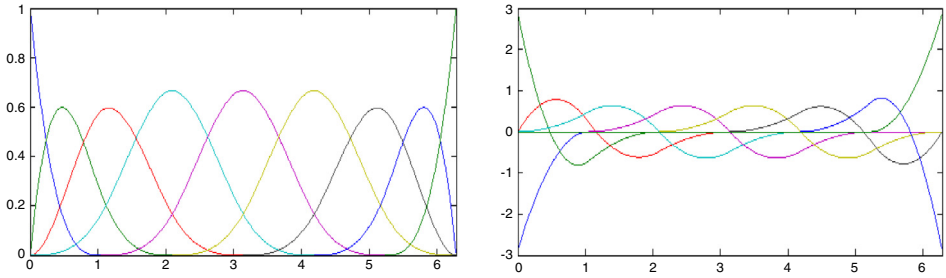

Fig. 66. **Left:** Cubic B-spline basis functions with open knot vector. **Right:** The corresponding quadratic derivatives of cubic B-spline basis functions.

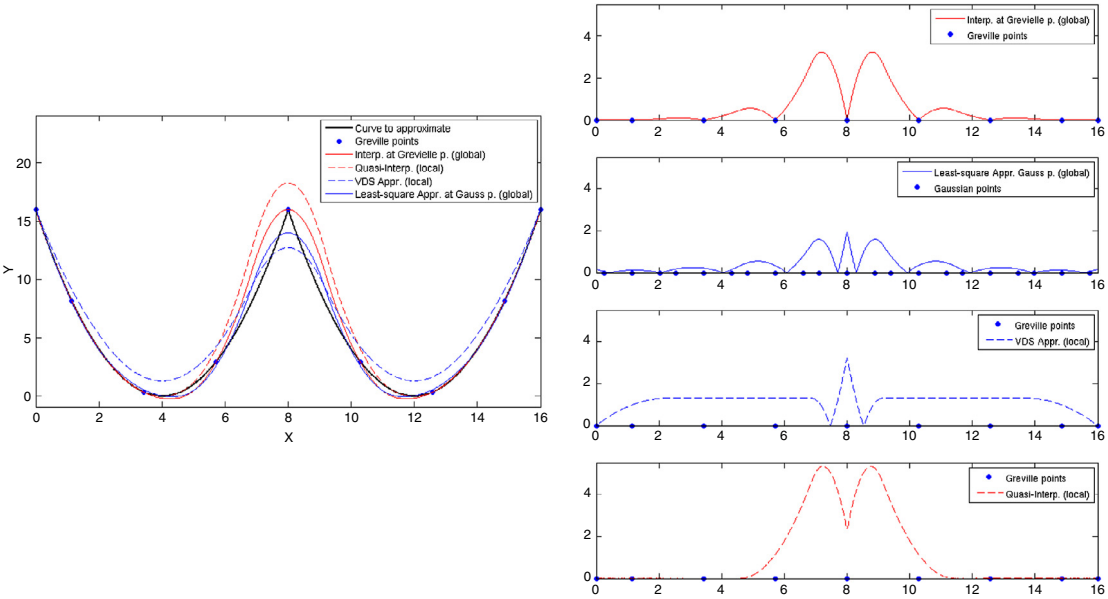

Fig. 67. **Left:** Quadratic function interpolated and approximated using four different methods. **Right:** Exact relative difference error (RDE): Global spline interpolation at Greville points (RDE $= 0.1425$); least-squares approximation at $n = 3$ Gaussian points (RDE $= 0.0761$); VDSA at Greville points (RDE $= 0.1613$); quasi-interpolation with 3 point quadratic quasi-interpolant at Greville points (RDE $= 0.2381$).

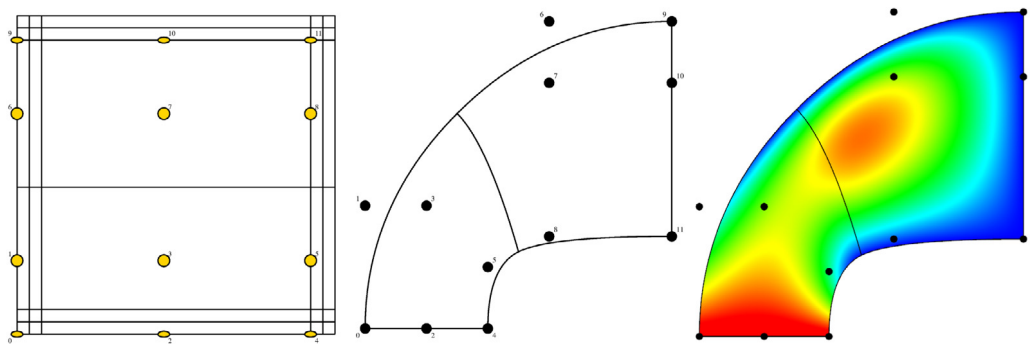

Fig. 68. Tensor product B-spline mesh for a quadratic 2D problem. **Left:** Knot vectors Ξ and $\mathcal{H}$ for the first and second parameter direction define the knot lines forming the tensor product mesh within the parametric space $(\Xi, \mathcal{H})$ with respective basis functions (yellow). **Center:** Respective mesh with control points in physical space. **Right:** Solution to the classical Poisson equation. Details of the knot vectors can be found in the text below. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

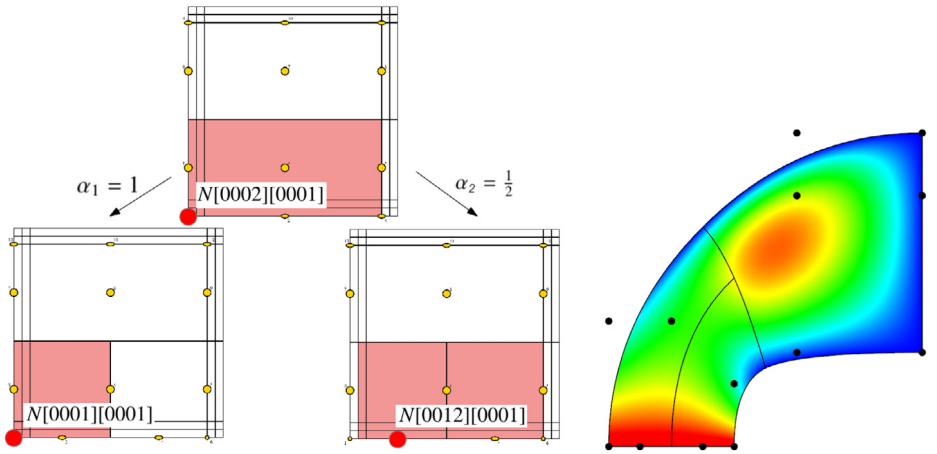

Fig. 69. Local mesh refinement (cf. Eq. (96)). **Left:** The basis function $N_{\Xi} = N[0002][0001]$ is split into two new basis functions $N_{\Xi_1}(\xi) = N[0001][0001]$ and $N_{\Xi_2}(\xi) = N[0012][0001]$ (red). **Right:** For the new refined geometry more accurate analysis results are computed. Poisson solution on the refined geometry with new control point distribution.

This implies that one LR B-spline basis function is split into two either already known and/or new LR B-spline basis functions $N_{\Xi_1}(\xi)$ and $N_{\Xi_2}(\xi)$:

$$N_{\Xi}(\xi) = \alpha_1 N_{\Xi_1}(\xi) + \alpha_2 N_{\Xi_2}(\xi), \tag{93}$$

where

$$\alpha_1 = \begin{cases} 1 & \xi_{p+1} \leq \hat{\xi} \leq \xi_{p+2}, \\ \dfrac{\hat{\xi} - \xi_1}{\xi_{p+1} - \xi_1} & \xi_1 \leq \hat{\xi} \leq \xi_{p+1} \end{cases} \tag{94}$$

and

$$\alpha_2 = \begin{cases} \dfrac{\xi_{p+2} - \hat{\xi}}{\xi_{p+2} - \xi_2} & \xi_2 \leq \hat{\xi} \leq \xi_{p+2}, \\ 1 & \xi_1 \leq \hat{\xi} \leq \xi_2. \end{cases} \tag{95}$$

If we proceed with the example shown in Fig. 68 the knot $\hat{\xi} = 1$ is inserted into the local knot vector $\Xi = [0002]$. We generate two B-splines with local knot vectors $\Xi_1 = [0001]$ and $\Xi_2 = [0012]$ with α-values as follows

$$N[0002][0001] = 1\ N[0001][0001] + \frac{1}{2}\ N[0012][0001]. \tag{96}$$

This implies that the basis function $N_{(p,\Xi;q\mathcal{H})}(\xi, \eta) = N[0002][0001]$ is replaced by two new basis functions $N_{(2,\Xi_1;2,\mathcal{H}_1)}(\xi, \eta) = N[0001][0001]$ and $_{(2,\Xi_2;2,\mathcal{H}_2)}(\xi, \eta) = N[0012][0001]$. The mesh is locally refined consisting now of three elements and a t-junction is created as shown in Fig. 69.

References

[1] T.J.R. Hughes, J.A. Cottrell, Y. Bazilevs, Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement, Comput. Methods Appl. Mech. Engrg. 194 (2005) 4135–4195.
[2] T. Dokken, T. Lyche, K. Pettersen, Polynomial splines over locally refined box-partitions, Comput. Aided Geom. Design 30 (3) (2013) 331–356.
[3] K.A. Johannessen, T. Kvamsdal, T. Dokken, Isogeometric analysis using LR B-splines, Comput. Methods Appl. Mech. Engrg. 269 (2014) 471–514.
[4] T.W. Sederberg, D.L. Cardon, G.T. Finnigan, N.S. North, J. Zheng, T. Lyche, T-spline simplification and local refinement, ACM Trans. Graph. 23 (2004) 276–283.
[5] T.W. Sederberg, J. Zheng, A. Bakenov, A. Nasri, T-splines and T-NURCCs, ACM Trans. Graph. 22 (3) (2003) 477–484.

[6] A.-V. Vuong, C. Giannelli, B. Jüttler, B. Simeon, A hierarchical approach to adaptive local refinement in isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 200 (49–52) (2011) 3554–3567.
[7] D. Schillinger, L. Dede, M.A. Scott, J.A. Evans, M.J. Borden, E. Rank, T.J. Hughes, An isogeometric design-through-analysis methodology based on adaptive hierarchical refinement of NURBS, immersed boundary methods, and T-spline CAD surfaces, Comput. Methods Appl. Mech. Engrg. 249–252 (2012) 116–150.
[8] M.J. Borden, M.A. Scott, J.A. Evans, T.J.R. Hughes, Isogeometric finite element data structures based on Bézier extraction of NURBS, Internat. J. Numer. Methods Engrg. 87 (1–5) (2011) 15–47.
[9] M.J. Borden, M.A. Scott, J.A. Evans, T.J.R. Hughes, Isogeometric finite element data structures based on Bézier extraction of T-splines, Internat. J. Numer. Methods Engrg. 88 (2) (2011) 1–40.
[10] W. Wang, Y. Zhang, M.A. Scott, T.J.R. Hughes, Converting an unstructured quadrilateral mesh to a standard T-spline surface, Comput. Mech. 48 (2011) 477–498.
[11] J.E. Marsden, T.J.R. Hughes, Mathematical Foundations of Elasticity, Dover, New York, 1994.
[12] L. Piegl, W. Tiller, The NURBS Book, Springer-Verlag, Berlin, Heidelberg, New York, 1997.
[13] T. Lyche, K. Mørken, Spline methods, Spline metoder, Lecture Notes, INF-MAT5340, Spring 2011 (Aril 5 2011).
[14] T. Greville, Theory and Applications of Spline Functions, Academic Press, New York, 1969.
[15] A. Stroud, D. Secrest, Gaussian quadrature formulas, in: Series in Automatic Computation, Prentice-Hall, 1966.
[16] R.L. Taylor, Isogeometric analysis of solids & structures – small and finite deformation applications, in: ECCOMAS Special Interest Conference – TCCM2011 Trends & Challenges in Computational Mechanics, Padua, Italy, Invited lecture (September 12–14 2011).
[17] J.A. Cottrell, T.J.R. Hughes, Y. Bazilevs, Isogeometric Analysis: Toward Integration of CAD and FEA, John Wiley & Sons, 2009.
[18] T. Kanai, Y. Yasui, Surface quality assessment of subdivision surfaces on programmable graphics hardware, in: Proc. International Conference on Shape Modeling and Applications 2004, IEEE CS Press, 2004, pp. 129–136.
[19] J.M. Hjelmervik, Heterogeneous Computing with Focus on Mechanical Engineering, (Ph.D. Thesis), Oslo, 2009.
[20] W.E. Lorensen, H.E. Cline, Marching cubes: A high resolution 3D surface construction algorithm, Comput. Graph. 21 (4) (1987) 163–169.
[21] H. Müller, M. Stark, Adaptive generation of surfaces in volume data, Vis. Comput. 9 (4) (1993) 182–199.
[22] P. Cignoni, L. De Floriani, C. Montani, E. Puppo, R. Scopigno, Multiresolution modeling and visualization of volume data based on simplicial complexes, in: Proceedings of the 1994 Symposium on Volume Visualization, VVS '94, ACM, New York, NY, USA, 1994, pp. 19–26.
[23] T. Martin, E. Cohen, M.M. Kirby, Direct isosurface visualization of hex-based high-order geometry and attribute representations, IEEE Trans. Vis. Comput. Graphics 18 (5) (2012) 753–766.
[24] A. Knoll, I. Wald, S. Parker, C. Hansen, Interactive isosurface ray tracing of large octree volumes, in: Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing, 2006, pp. 115–124.
[25] M. Levoy, Volume rendering: display of surfaces from volume data, Comput. Graph. Appl. (1988).
[26] M. Geimer, O. Abert, Interactive Ray Tracing of Trimmed Bicubic Bézier Surfaces without Triangulation, in: In Proceedings of WSCG, 2005, pp. 71–78.
[27] H.-F. Pabst, J. Springer, A. Schollmeyer, R. Lenhardt, C. Lessig, B. Froehlich, Ray casting of trimmed NURBS surfaces on the GPU, in: Symposium on Interactive Ray Tracing, 2006, pp. 151–160.
[28] M. Nießner, C. Loop, M. Meyer, T. Derose, Feature-adaptive GPU rendering of Catmull–Clark subdivision surfaces, ACM Trans. Graph. 31 (1) (2012) 6:1–6:11.
[29] R. Wright, B. Lipchak, N. Haemel, OpenGL Superbible: Comprehensive Tutorial and Reference, fourth ed., Addison-Wesley Professional, 2007.
[30] R.J. Rost, OpenGL Shading Language, second ed., Addison-Wesley Professional, 2005.
[31] P.L. Gould, Introduction to Linear Elasticity, Springer-Verlag, 1999.
[32] M. Kumar, T. Kvamsdal, K.A. Johannessen, Superconvergent patch recovery and a posteriori error estimation technique in adaptive isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 316 (2017) 1086–1156.
[33] S. Govindjee, J. Strain, T.J. Mitchell, R.L. Taylor, Convergence of an efficient local least-squares fitting method for bases with compact support, Comput. Methods Appl. Mech. Engrg. 213–216 (2012) 84–92.
[34] K.M. Mathisen, K.M. Okstad, T. Kvamsdal, S.B. Raknes, Isogeometric analysis of finite deformation nearly incompressible solids, J. Struct. Mech. 44 (3) (2011) 260–278.
[35] K.M. Mathisen, K.M. Okstad, T. Kvamsdal, S.B. Raknes, Simulation of contact between subsea pipeline and trawl gear using mortar-based isogeometric analysis, in: F. Salvatore, R. Broglia, R. Muscari (Eds.), Proceedings of the VI International Conference on Computational Methods in Marine Engineering, MARINE 2015, Rome, Italy, 2015, pp. 290–305.
[36] M. Kristoffersen, T. Børvik, M. Langseth, O.S. Hopperstad, H. Ilstad, E. Levold, Damage and failure in an X65 steel pipeline caused by trawl gear impact, in: ASME 2013: 32nd International Conference on Ocean, Offshore and Arctic Engineering, Vol. 2B: Structures, Safety and Reliability, ASME Press, 2013, pp. 1–10.
[37] M. Kristoffersen, T. Børvik, I. Westermann, M. Langseth, O.S. Hopperstad, Impact against X65 steel pipes – An experimental investigation, Int. J. Solids Struct. 50 (20–21) (2013) 3430–3445.
[38] M. Kristoffersen, F. Casadei, T. Børvik, M. Langseth, G. Solomos, O.S. Hopperstad, Numerical simulations of submerged and pressurised X65 steel pipes, in: Computational Plasticity XII Fundamentals and Applications, Barcelona, Spain, 3–5 September 2013, International Center for Numerical Methods in Engineering, CIMNE, 2013, pp. 1384–1395.
[39] S. Lipton, J. Evans, Y. Bazilevs, T. Elguedj, T. Hughes, Robustness of isogeometric structural discretizations under severe mesh distortion, Comput. Methods Appl. Mech. Engrg. 199 (5–8) (2010) 357–373.
[40] T. Kvamsdal, K.M. Okstad, Integrated Computer Aided Design and Analysis (ICADA), Vitality through locally refined splines and immersed boundary techniques, in: A. Logg, K.A. Mardal (Eds.), Proceedings of the 26th Nordic Seminar on Computational Mechanics, Oslo, Norway, 2013, pp. 191–194.

[41] T. Kvamsdal, K.M. Okstad, K.A. Johannessen, M. Kumar, Immersed and adaptive Isogeometric analysis of thin plates, in: Presentation at 5th International Conference on Computational Methods in Marine Engineering, Hamburg, Germany, May 2013, 2013.
[42] N. Schubert, I. Scholl, Comparing GPU-based multi-volume ray casting techniques, Comput. Sci. 26 (1–2) (2011) 39–50.
[43] S.N. Bernstein, Démonstration du Théorème de Weierstrass fondée sur le calcul des Probabilités, Comm. Soc. Math. Kharkov 2. Series XIII (1) (1912) 1–2.
[44] W. Boehm, Inserting new knots into B-spline curves, Comput.-Aided Des. 12 (4) (1980) 199–201.
[45] M. Kumar, T. Kvamsdal, K.A. Johannessen, A simple a posteriori error estimation in adaptive isogeometric analysis, Comput. Math. Appl. 70 (7) (2015) 1555–1582.
[46] K.A. Johannessen, M. Kumar, T. Kvamsdal, Divergence-conforming discretization for Stokes problem on locally refined meshes using LR B-splines, Comput. Methods Appl. Mech. Engrg. 293 (2015) 38–70.
[47] Y.W. Bekele, T. Kvamsdal, A.M. Kvarving, S. Nordal, Adaptive isogeometric finite element analysis of steady-state groundwater flow, Int. J. Numer. Anal. Methods Geomech. 40(5) (2016) 738–765.
[48] K.A. Johannessen, F. Remonato, T. Kvamsdal, On the similarities and differences between Classical Hierarchical, Truncated Hierarchical and LR B-splines, Comput. Methods Appl. Mech. Engrg. 291 (2015) 64–101.