Technische Universität München
TUM School of Engineering and Design

Space-time multi-level *hp*-finite elements for heat evolution in laser powder bed fusion additive manufacturing

Philipp Michael Kopp, M.Sc. (hons)

Vollständiger Abdruck der von der TUM School of Engineering and Design der Technischen Universität München zur Erlangung eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz:		Prof. Dr.-Ing. habil. Stefan Kollmannsberger
Prüfer der Dissertation:	1.	Prof. Dr. rer. nat. Ernst Rank
	2.	Prof. Dr. Victor Calo
	3.	Prof. Dr.-Ing. habil. Fabian Duddeck

Die Dissertation wurde am 21.08.2023 bei der Technischen Universität München eingereicht und durch die TUM School of Engineering and Design am 13.11.2023 angenommen.

Contents

Zusammenfassung

Das Laser-Pulverbettfusions-Verfahren ist eine additive Fertigungstechnologie für den Druck dreidimensionaler Metallobjekte. Es bietet im Vergleich zu herkömmlichen Verfahren mehr Gestaltungsfreiheit und oft geringere Herstellungskosten. Dabei bleiben einige Hürden zu überwinden, die einen konsistenten automatischen Druckprozess verhindern. So können sich beispielsweise Residualspannungen ansammeln, die dann unerwünschte Verformungen verursachen. Übermäßiges oder fehlendes Schmelzen kann zu erheblichen Abweichungen vom geometrischen Modell führen und bestimmte Abkühlungsmuster können eine inkonsistente oder unerwünschte Mikrostruktur bilden. Computersimulationen sind hier unerlässlich, um die treibenden physikalischen Phänomene zu verstehen, geeignetere Prozessparameter zu finden und damit die Qualität der gedruckten Objekte zu optimieren. Thermische Modelle basierend auf der Wärmeleitungsgleichung sind dabei aufgrund des vergleichsweise niedrigen Berechnungsaufwandes attraktiv, da sie die Simulation längerer Zeiträume ermöglichen und dennoch so viel Einblick in den Prozess geben, dass problematische Bereiche identifiziert werden können.

Klassische Ansätze zur Approximation der Wärmeleitungsgleichung verwenden Finite Elemente zur Diskretisierung der räumlichen Dimension in Kombination mit einem Zeitschrittschema. Insbesondere *hp*-Finite Elemente können die Mehrskaligkeit der Lösung erfassen, indem sie das Netz in Richtung Laserpunkt verfeinern und gleichzeitig den Polynomgrad erhöhen um glatte Temperaturverläufe in der Peripherie darzustellen. Bei der Verwendung von Zeitschrittverfahren für das Laser-Pulverbettfusions-Verfahren gibt es jedoch zwei große Herausforderungen. Erstens wird die genaue zeitliche Auflösung, die im Umfeld des Schmelzbads erforderlich ist, an anderer Stelle nicht benötigt, kann aber dort erhebliche Rechenressourcen verbrauchen. Zweitens müssen einzelne Zeitschritte in einer Simulation des gesamten Prozesses in potenziell unter einer Millisekunde berechnet werden. Zwar können effiziente räumliche Diskretisierungen einzelne Zeitschritte in unter einer Sekunde berechnen, jedoch erfordert eine weitere Reduktion um mehrere Größenordnungen eine massiv parallele Implementierung. Bei solchen kleinen Problemgrößen dominiert allerdings der Kommunikationsaufwand schnell und beschränkt somit eine Skalierung mit der Anzahl an Rechenknoten.

In dieser Arbeit wird eine alternative Methode zur Simulation der Temperaturentwicklung im Laser-Pulverbettfusions-Verfahren vorgestellt, die auf einer Raum-Zeit-Diskretisierung mit Finiten Elementen basiert. Hierbei ist die Zeit die vierte Dimension und die Finite-Elemente-Netze und -Basisfunktionen hängen nicht nur von den drei räumlichen Dimensionen, sondern auch von der Zeit ab. Durch die Wahl eines groben Netzes mit einer lokalen Verfeinerung in vier Dimensionen in Richtung Laserpunkt wird die zeitliche Genauigkeit direkt an die räumliche Genauigkeit gekoppelt. Die Simulation wird zusätzlich in aufeinanderfolgende Zeitabschnitte mit kontrollierbarer Dauer aufgeteilt, wodurch die Problemgröße erhöht werden kann, um alle potenziell verfügbaren Rechenressourcen in Anspruch nehmen zu können.

Um die Vorteile von *hp*-Methoden in einer Raum-Zeit-Diskretisierung zu nutzen, ist eine effiziente Methode zur Konstruktion von *hp*-Basen auf vierdimensionalen Netzen erforderlich. Die kürzlich eingeführte Multi-Level *hp*-Methode verwendet einen Überlagerungsansatz mit einem einfacheren Regelwerk im Gegensatz zu klassischen Verfeinerungsstrategien, in denen die zu verfeinernden Elemente ersetzt werden. Dabei wird eine objektorientierte Datenstruktur aufgebaut, die es ermöglicht, die passenden Formfunktionen zu filtern und zu verbinden. Ursprünglich vorgestellt für maximal drei Dimensionen, nimmt die Komplexität der Datenstruk-

tur in vier oder mehr Dimensionen dramatisch zu und effiziente Implementierungen werden zunehmend schwieriger. Mit dieser Arbeit wird eine datenorientierte Alternative eingeführt, die nur grundlegende Nachbarschaftsinformationen zwischen den Zellen des hierarchisch verfeinerten Netzes benötigt. Die hierzu entwickelten Algorithmen nutzen die reduzierte Menge an gespeicherten Informationen, um eine *hp*-Basis zu konstruieren, die äquivalent zur ursprünglichen Multi-Level *hp*-Methode ist und deren Einfachheit beibehält.

Dieser Ansatz wird anschließend verwendet, um eine Wärmegleichung mit temperaturabhängigen Koeffizienten zu diskretisieren, einschließlich eines Modells der scheinbaren Wärmekapazität (apparent heat capacity) zur Berücksichtigung der latenten Schmelzenthalpie. Eine volumetrische Wärmequelle ermöglicht es dem Laser, im thermischen Modell in die Metalloberfläche einzudringen, wodurch dessen Gültigkeit in den Übergangsbereich zwischen Konduktions- und Keyhole-Modus erweitert wird. Die verwendete Raum-Zeit-Finite-Elemente-Formulierung testet mit den zeitlichen Ableitungen der Ansatzfunktionen. Dies führt zu einer optimalen Methode, die es auch ermöglicht, die Simulation in aufeinanderfolgende Zeitabschnitte aufzuteilen. Die Einführung einer auf das Laser-Pulverbettfusions-Verfahren zugeschnittenen Netzverfeinerungsstrategie, die den unmittelbar vorhergehenden Verlauf des Laserpfades berücksichtigt, ermöglicht eine Netzverfeinerung ohne adaptive Schleife.

Die Implementierung der vorgestellten *hp*-Methode wird anhand eines linearen Poisson-Problems mit einer Punktsingularität verifiziert. Die Konvergenz des Energiefehlers ist exponentiell in Bezug auf die Anzahl der Unbekannten und die Laufzeit bleibt polynomial, was mit den hergeleiteten Abschätzungen übereinstimmt. Anschließend wird der AMB2018-02-Benchmark mit dem vorgestellten Raum-Zeit-Ansatz berechnet, bei dem die Modellparameter so bestimmt werden, dass die Abmessungen des Schmelzbades mit den experimentellen Daten übereinstimmen. Die Absorptivität und die Eindringtiefe werden benutzt, um die Breite und Tiefe des Schmelzbades anzupassen und die Regularisierung des Scheinbare-Wärmekapazität-Modells steuert dessen Länge. Unter Verwendung derselben Konfiguration wird danach ein Quadrat mit einer Seitenlänge von einem Zentimeter auf der Oberfläche einer Metallplatte schraffiert, um die Möglichkeit längerer Simulationen zu demonstrieren. Abschließend zeigt ein Vergleich zwischen der Raum-Zeit-Diskretisierung und einem Zeitschrittschema mit lokaler *hp*-Verfeinerung in drei Dimensionen eine deutliche Beschleunigung des Berechnungsprozesses zugunsten der vierdimensionalen Raum-Zeit-Methode bei hohen peripheren Berechnungskosten. In diesem Fall überwiegt der Vorteil einer gröberen peripheren zeitlichen Auflösung die erhöhten Kosten, die sich aus der verstärkten Kopplung der Basisfunktionen und der direkten Lösung des unsymmetrischen Gleichungssystems ergeben.

Abstract

Laser powder bed fusion is an additive manufacturing technology for printing three-dimensional metal objects that offers more design freedom and often reduces manufacturing costs over conventional approaches. There remain several challenges that prevent a consistent automatic printing process. For example, residual stresses may accumulate and cause unwanted deformations, excessive or lack of fusion can lead to significant deviations from the geometric model, and specific cooling patterns may form an inconsistent or undesired microstructure. Computer simulations are essential for understanding the driving physical phenomena, and they can help improve the part quality by finding better process parameters. Thermal models based on the heat equation are attractive due to their low computational cost, which enables the simulation of longer durations while still giving some insight into the process and allowing to identify problematic areas.

Classical approaches to discretize the heat equation use finite elements for the spatial dimension in combination with a time-stepping scheme. In particular, *hp*-finite elements can resolve the multi-scale nature of the solution by refining the mesh towards the laser spot while also allowing to elevate the polynomial degree to approximate the smooth temperature in the periphery. Two major challenges exist when using time-stepping schemes for laser powder bed fusion processes. First, the accurate temporal resolution required around the melt pool is not needed elsewhere and may consume significant computational resources. Second, the long duration of part-scale simulations may require computing single time steps in even less than one millisecond. While efficient spatial discretizations can reduce the computational time of a single time step below one second, their massively parallel implementation necessary to further reduce the computational time is limited by the communication overhead that starts to dominate when distributing individual time step meshes.

This thesis introduces an alternative way of simulating the temperature evolution in laser powder bed fusion based on a space-time finite element discretization. In this setting, time is the fourth dimension, and the finite element meshes and basis functions depend not only on the three spatial dimensions but also on time. By choosing a coarse mesh and locally refining the mesh in four dimensions towards the laser spot, the temporal accuracy is directly coupled to the spatial accuracy. The simulation is split into consecutive time slabs with a controllable duration, which allows increasing the problem sizes to utilize the available computational resources optimally.

Exploiting the advantages of *hp*-methods in space-time discretizations requires an efficient construction of *hp*-bases on four-dimensional meshes. The recently introduced multi-level *hp*-method uses a *refine-by-superposition* approach with a simpler ruleset than classical *refine-by-replacement* strategies. It builds on an object-oriented data structure that helps to filter and connect matching shape functions and was initially presented for one, two, and three dimensions. The complexity of the data structure grows dramatically in four or more dimensions, and efficient implementations become increasingly difficult. This thesis introduces a data-oriented alternative that only requires basic adjacency information between the cells of the hierarchically refined mesh. The presented algorithms reduce the amount of stored information needed to construct an *hp*-basis equivalent to the original multi-level *hp*-method while maintaining its simplicity.

This approach is then used to discretize a heat equation with temperature-dependent coefficients, including an apparent heat capacity model to account for the latent heat of fusion.

A volumetric heat source allows the laser to penetrate the metal surface within the thermal model, extending its range of validity into the transition zone between conduction and keyhole mode. The space-time finite element formulation uses the time derivative of the trial functions as test functions, leading to an optimal method that allows separating the simulation into consecutive time slabs. Introducing a mesh refinement strategy tailored to laser powder bed fusion processes enables mesh refinements without an adaptive loop by considering the recent history of the laser path.

The *hp*-refinement implementation is verified on a linear Poisson problem with a point singularity. The convergence of the energy error is exponential with respect to the number of unknowns, and the runtime stays polynomial, which agrees with the derived estimates. Then, the AMB2018-02 benchmark is computed using the presented space-time approach, where the model parameters are tuned to match the melt pool dimensions to the experimental data. The absorptivity and the penetration depth are adjusted to fit the width and depth of the melt pool, and the regularization of the apparent heat capacity controls its length. The same setup is used to hatch a square with one centimeter side length on the surface of a metal plate to demonstrate the possibility of longer simulations. Finally, a comparison of the space-time discretization to a time-stepping scheme with local *hp*-refinement in three dimensions shows significant speedups in favor of the space-time method when the peripheral computational cost is high. In this case, the benefit of adapting the temporal accuracy outweighs the increased cost resulting from the greater coupling of basis functions and the direct solution of the unsymmetric equation system.

Acknowledgments

The work of this thesis was conducted during my time at the Chair for Computation in Engineering and the Chair of Computational Modeling and Simulation at the Technical University of Munich. It was funded in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) through grant KO 4570/2-1, for which I am grateful. Many people I met during this time shaped my path in one way or another or were just fun to be around.

First, I want to thank Professor Rank for allowing me to work and graduate in his team. He trusted me after pitching the topic of this thesis to him and encouraged me to look into it, although we already had a different topic in mind. The place and the people always felt special to me and when asking why, one must first look to the top. Professor Rank has consistently put great effort into creating an environment for his team that provides much freedom for creativity while giving us the feeling that he has our back should there be any trouble. He was always available to give great advice and continues to do so today. Since I can't imagine Professor Rank ever truly retiring, I'm confident that we will see more of him in the future.

Stefan Kollmannsberger equally contributed to the specialness of our group. Most of the time, he puts his team before himself, enabling the people around him and setting them up for success. This selflessness may not be very visible or shiny to the outside world, but it deserves every possible credit. Still, he has accumulated an impressive academic record for which he was finally rewarded recently, which makes me very happy. Of course, Stefan is a great person to discuss science and life with. He often sees the bigger picture and helps to break out of tunnel vision, which is sometimes easy to get stuck in. I'm sure we will continue to work on many things together.

Victor Calo and I first met during HOFEIM 2014 at Frauenchiemsee, where I was a Bachelor's student helping with the conference. Victor spent a lot of time and effort sharing his experience with me, which was valuable in many ways. His view on science, particularly our research field, which we discussed quite extensively during my visit to Perth in 2016, has been very healthy for me since. He also helped me improve my scientific writing skills and understand mathematical concepts. The discussions with him have always been great, and I hope we continue this kind of exchange.

I also want to thank Professor Duddeck for agreeing to be the third reviewer and taking the time to thoroughly read through my thesis. I know time is rare as a Professor, and I appreciate the effort and helpful advice very much. Maybe there is a way to work on something together in the future?

There are many other people that I want to mention. It all started with Tino Bog asking me during an oral exam in 2012 whether I would be interested in a student job. He promised they'd do a lot of object-oriented C++, which turned out to be very true. Later, I worked for Nils Zander, who taught me how to do science and really set me up for the path I followed since. Nils and Tino not only spent a lot of time and effort designing their new finite element code but also made sure that the growing team of developers could learn from them as much as possible. Over the years, Nina Korshunova, John Jomo, Lisa Hug, Ali Özcan, László Kudela, Mohamed Elhaddad, Davide D'Angella, Alex Paolini, Oguz Oztoprak, and Benjamin Wassermann joined. Chris Ertl and Nevena Perovic didn't work on our code but were equally part of the team. There are so many awesome memories, like the winter seminars every year, the conferences and workshops we visited together, the regular coffee breaks with Nina and our "jungle" office, or the "science gone wrong" and other nerdy memes with László and Mohamed.

I also want to thank Hanne Cornils and Simon Vilgertshofer, with whom I shared some of the administrative burdens.

After Covid then started a new chapter where most of the old team had graduated, and Tim Bürchner, Vijay Holla, Leon Herrmann, Alireza Daneshyar, and Divya Singh joined. The frequent discussions with Tim and Vijay, who I work with quite closely, are always something I very much look forward to. Outside of TUM, we often met Lars Radtke, Andre Hildebrandt-Raj, and Wadhah Garhuom from Hamburg and Massimo Carraturo from Pavia, who I am happy to see again every time. Even further away, but still somehow part of my extended academic world, are Philipp Bucher, Ege Gümüs, and Andrew Brodie. We met during our Master's studies and have shared many of the struggles since. I am also very thankful for the continuous and loving support of my parents. Especially in recent years, our frequent coffees often brought clarity that I didn't have before.

Philipp Kopp
München, April 2024

Chapter 1

Introduction

1.1 Laser powder bed fusion of metals

Additive manufacturing (AM), also called 3D printing, is a family of methods for automatically producing three-dimensional structures. Unlike subtractive approaches that remove material during manufacturing (such as milling), additive methods successively add material to build the final product. This is typically done in a layerwise fashion, where on each layer, new material is added or consolidated selectively on areas that are inside of the designed structure. As a result, the parts printed using AM techniques are limited in terms of feature size and surface roughness. Before the manufacturing starts, a geometric model must be prepared in a computer-aided design (CAD) software, where support structures may be added to ensure the mechanical stability of the component and provide sufficient thermal dissipation. The three-dimensional geometric model is then subdivided into thin horizontal slices for layerwise manufacturing. The software constructs a path for each slice according to the process parameters that fills the areas inside the geometric model. The structure is then printed using this path, after which several postprocessing steps (e.g., removing support structures) might be necessary to obtain the final product.

The ISO 17296-2:2015(E) (2015) standard defines seven additive manufacturing process categories: vat photopolymerization, material jetting, binder jetting, powder bed fusion, material extrusion, directed energy deposition, and sheet lamination. These categories vary significantly in their application fields and used materials. Obtaining an overview of all available techniques can be time-consuming as various manufacturers sell their approaches under different names. For example, current entry-level consumer 3D printers often use fused filament fabrication (FFM), also called fused deposition modeling (FDM), where molten thermoplastic material is deposited through a nozzle. FFM/FDM falls in the category of material extrusion. Similar ideas are used in directed energy deposition (DED) methods for printing metal structures. Instead of directly depositing the material, the metal is supplied as cable or in powder form by an inert gas carrier and melted in place by a high-power laser, an electron beam, or a plasma arc. This technology allows printing large structures with aeronautics as one major field of application, where it is used to print rocket engine components (Blakey-Milner et al., 2021), for example. In construction, DED allows printing structural components or even entire bridges, such as the MX3D bridge in Amsterdam (Buchanan & Gardner, 2019). Powder bed fusion (PBF) methods are an alternative to DED for smaller structures that require more accuracy. Instead of a selective deposition, metal powder is added to the entire layer and then selectively melted by a laser (PBF-LB/M) or an electron beam (PBF-EB/M). Figure 1.1 shows a typical PBF-LB/M build chamber with an active laser. After finishing one layer,

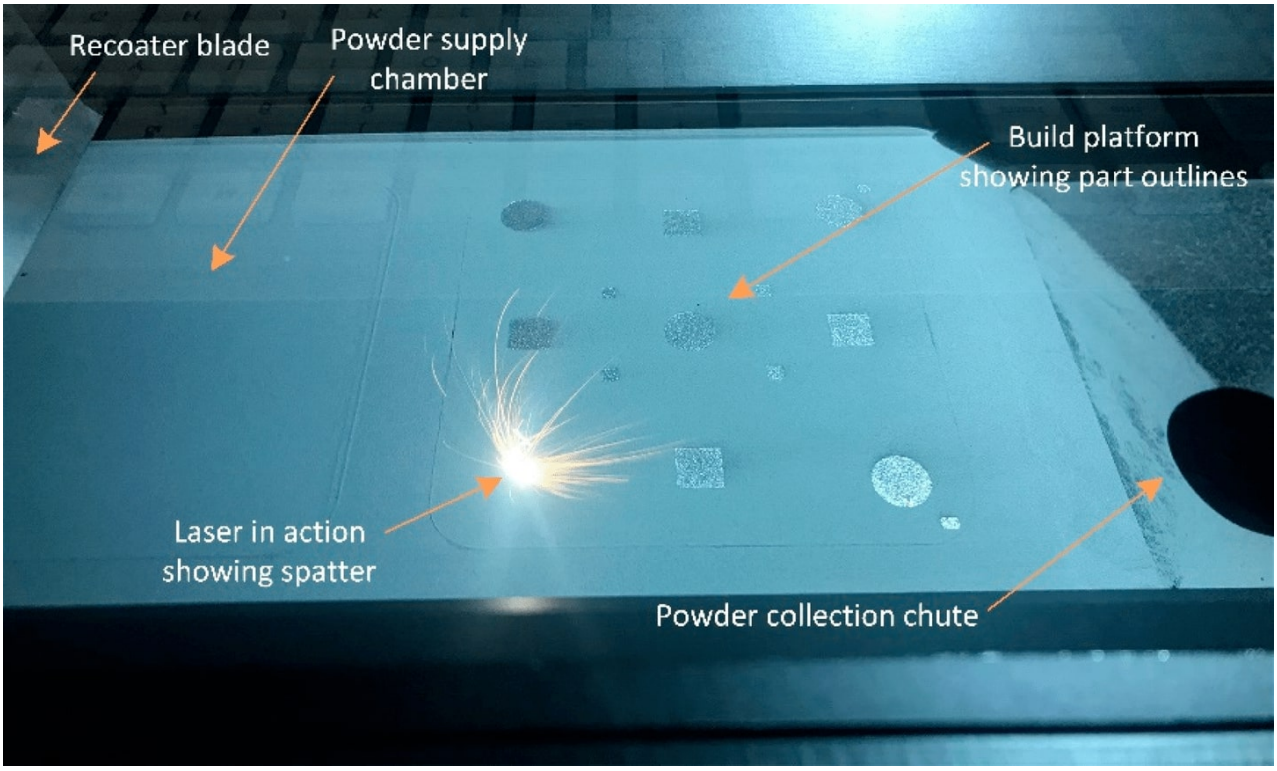


Figure 1.1: Elements of a PBF-LB/M build chamber. From Harkin et al. (2022), used under Creative Commons CC BY 4.0 license.

the build platform on the right moves downwards by one layer thickness, and the recoater blade on the left supplies new metal powder from the powder supply onto the platform with the partially built structure. Popular application fields of PBF methods besides aerospace engineering are the automotive and medical industries, where high accuracy for smaller parts is essential. Figure 1.2 shows a partially submerged titanium prosthesis manufactured with PBF-EB/M and cleaned afterwards. Powder bed fusion is also used with thermoplastic materials in PBF-LB/P and PBF-EB/P. While this thesis focuses on PBF-LB/M, many of its challenges exist in other PBF variants that may, therefore, equally benefit from the presented space-time simulation approach.

Printing parts with PBF is a complex process involving many physical phenomena over many spatial and temporal scales. Usually, only a small window of process parameters yields acceptable results, and even within this window, manufacturing failures and defects in the printed objects are common. Therefore, it is crucial to understand the process and develop computational models that allow optimizing the printing process based on the simulation results. A comprehensive discussion of current PBF-LB/M technology can be found in Yadroitsev et al. (2021). Current PBF-LB/M systems typically use a build chamber of 20 cm to 100 cm size with a build plate that supports the printed structure. On each new layer of metal powder, a high-power (typically infrared) laser scans the interior of the structure on the current intersection plane using the computer-generated path from the preprocessing stage. Figure 1.3 shows two adjacent hatch lines with a sufficient melt pool overlap to avoid pores between them. Afterwards, the build plate is moved downwards by one layer thickness, and a new layer of powder is introduced by the recoating system that ensures a well-leveled surface (us-

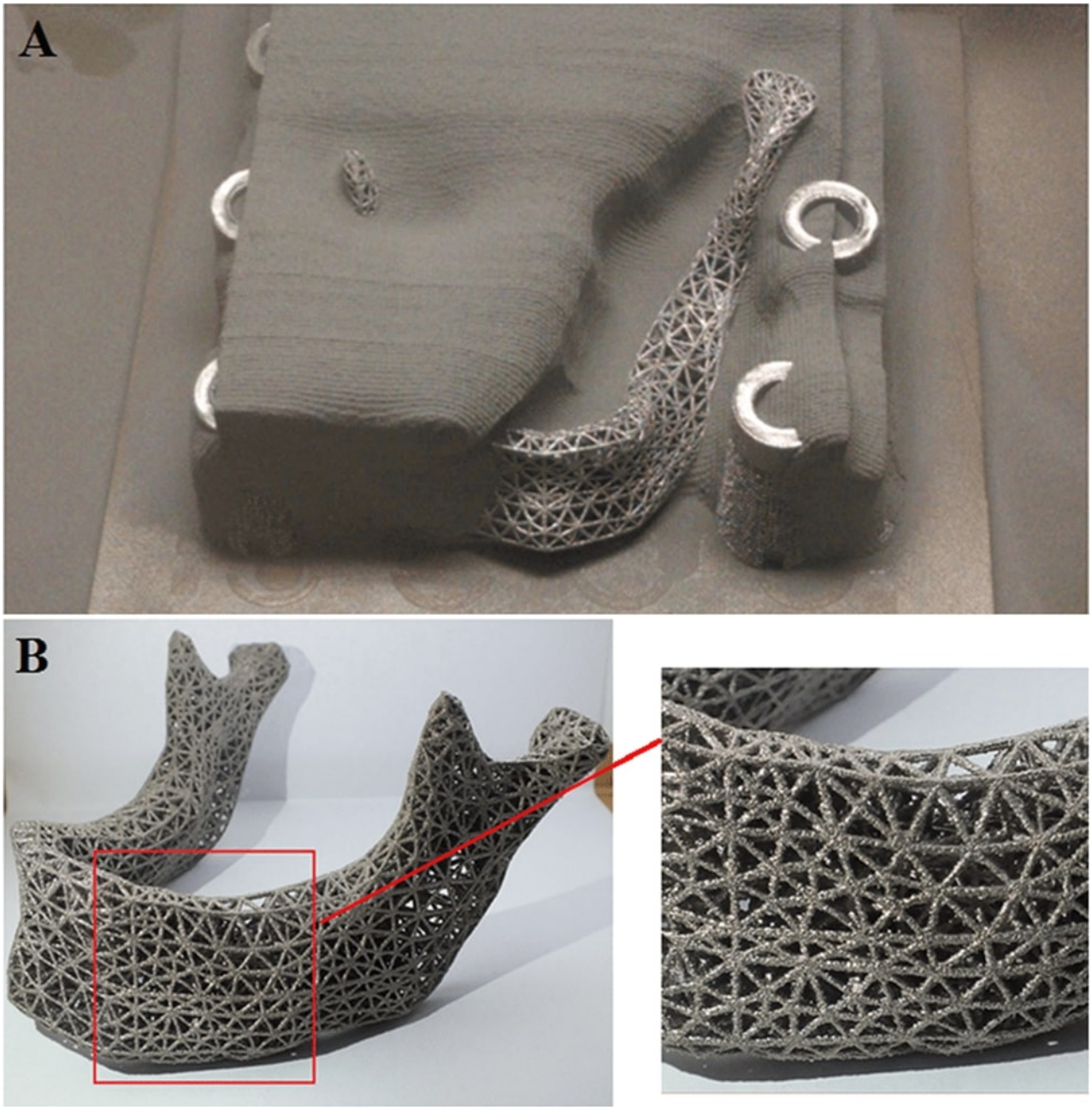


Figure 1.2: Removal of Ti6Al4V powder around a titanium prosthesis scaffold manufactured by PBF-EB/M. From Rongzeng et al. (2018), used under Creative Commons CC BY 4.0 license.

ing, for example, a leveling roller or blade). Moving the build plate has the advantage that the recoating system and the laser focus plane do not need to move vertically.

When the laser light hits the metal (powder) surface, it is partially reflected and absorbed, converting the radiant energy into heat. With enough laser power, the metal directly around and behind the laser spot is melted for a short time. Common laser powers range from below one hundred to a few hundred watts, but different powers can be necessary depending on the width and speed of the laser. The laser intensities in the focus plane are often Gaussian profiles

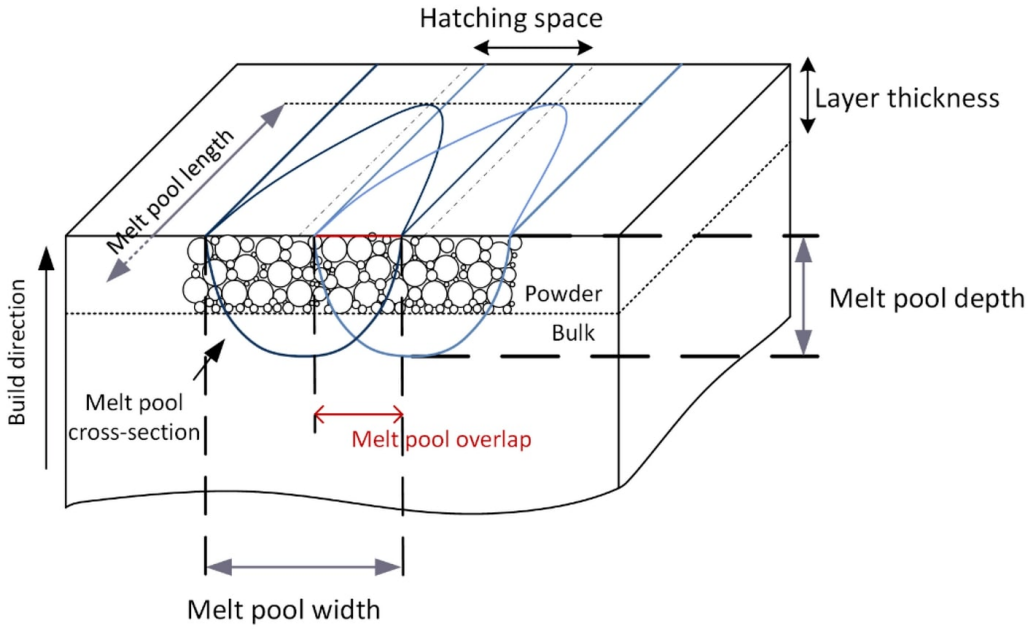


Figure 1.3: Areas on the current layer are hatched if they lie inside the geometric model. From Letenneur et al. (2017), used under Creative Commons CC BY 4.0 license.

with widths (4σ) of 60 µm to 250 µm. Typical laser velocities range from tens of centimeters per second to over one meter per second. After the laser passes, the metal cools down quickly and solidifies as the heat diffuses into the already-built structure. Figure 1.4 sketches the physical phenomena around a laser beam traveling from left to right. The metal powder surrounding the solidified object has a lower heat conductivity due to its porosity. Outside the melt pool, heat conduction and the induced mechanical response dominate; inside the melt pool, fluid flow and evaporation effects play an important role. The strong temperature gradients towards the laser spot lead to a rapid decrease in density, causing the liquid to expand under the laser spot. The strong density gradients also reduce the surface tension on the liquid-gas interface enough to cause strong Marangoni forces. These are tangential to the liquid surface and point towards regions of higher surface tension, driving the liquid on top of the melt pool away from the laser spot. As a result, circulating flows in the melt pool

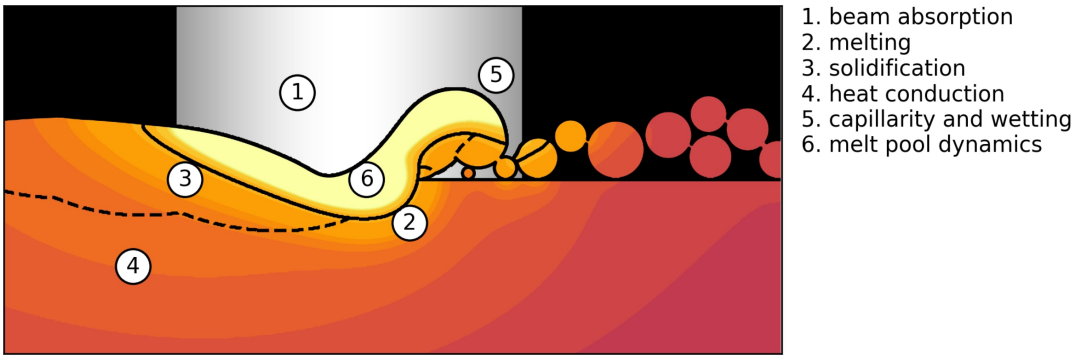


Figure 1.4: Physical phenomena in and around the melt pool. From Rausch et al. (2017), used under Creative Commons CC BY 4.0 license.

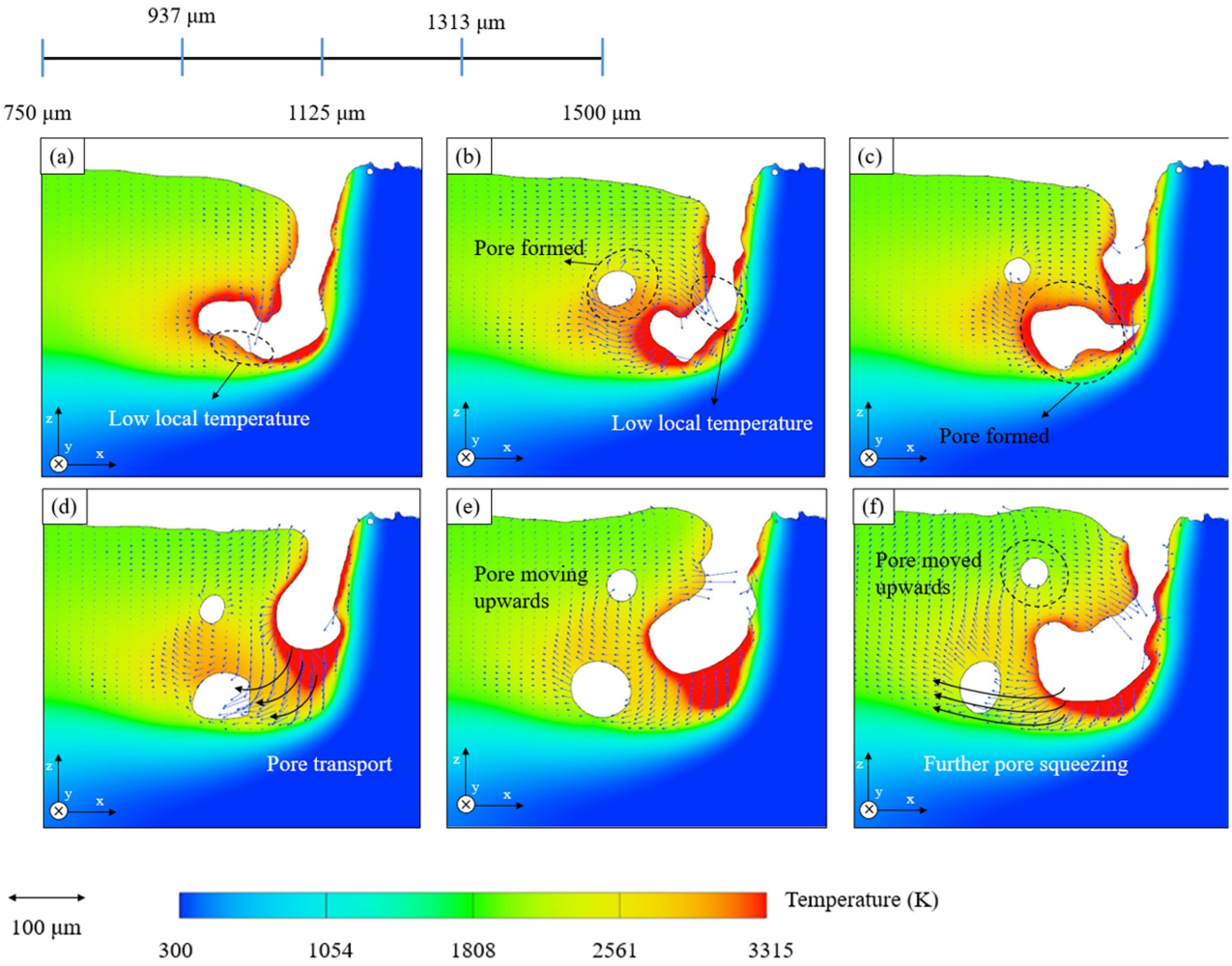


Figure 1.5: High laser powers evaporate metal and push the melt pool surface into the substrate due to recoil pressure. This change in geometry further increases the absorptivity and the melt pool depth, causing keyholes that leave unwanted pores when collapsing. From Bayat et al. (2019), used under Creative Commons CC BY-NC-ND 4.0 license.

significantly contribute to convective heat transport (see, e.g., Khairallah et al., 2016; Egorov et al., 2020). Together with the normal component of the surface tension, the Marangoni forces are one of the primary influences for the surface geometry of the melt pool. Additionally, with high enough laser power, the liquid metal starts to boil, and the evaporation cools the surface while pushing it downwards. This downward force, called recoil pressure, contributes to the formation of a depression in the melt pool. In extreme cases, the deep cavities that can form due to recoil pressure are called keyholes, and the process is said to be in keyhole mode (as opposed to conduction mode), as Figure 1.5 shows.

The highly complex local dynamics in PBF-LB/M processes give rise to many potential failures or defects in the final structure. Residual stresses occur when metal locally heats up quickly, expands and compresses nearby material, and cools down again while staying under tension. Depending on the laser path, these residual stresses can induce cracks or accumulate and deform the printed structure. Other defects in the built structure are pores that can originate from lack of fusion (LOF) or excessive heat. When the laser power density is too

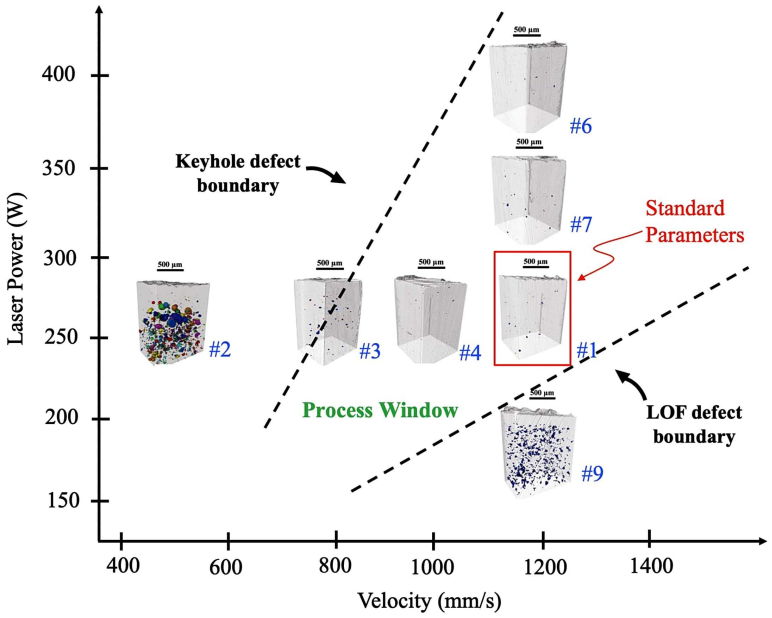


Figure 1.6: Window of feasible process parameters that result in sufficient fusion without formation of keyholes. From Gordon et al. (2020), used under Creative Commons CC BY-NC-ND 4.0 license.

low compared to the laser speed, the energy input onto the substrate surface is insufficient to fully melt the powder layer, resulting in residual pores due to partially melted grains. On the other hand, too much energy input can cause keyholes to periodically collapse with trapped air inside (Figure 1.5). Figure 1.6 shows the process window of parameters that balance these two extremes. However, the conditions during the manufacturing can vary greatly, depending on the part geometry and the selected laser pathing. Closely arranged laser paths and insolating surrounding powder can lead to significant local deviations from the optimal conditions within the process window. The resulting overheating zones can cause dimensional inaccuracies and porosity despite using parameters that perform well in an isolated setting.

1.2 The problem of scales in PBF-LB/M simulations

Computer simulations are essential for understanding and improving PBF-LB/M processes. The first step in setting up a simulation is choosing a mathematical model that describes the relevant phenomena in the process. For example, if one is interested in the coarse-scale thermal evolution in PBF, then the local effects in the melt pool do not need to be resolved as long as the energy balance is correct. The resulting partial differential equation (PDE) is defined on a spatial domain and usually enforces the conservation of quantities like energy, mass, or momentum. Solutions to such PDEs are functions in space and often time (e.g., the evolution of a temperature field) that are usually hard or impossible to obtain as analytical expressions; however, they can be approximated using numerical methods. These transform the continuously defined PDE into a discrete system of equations that can be solved on a computer and are called discretization methods. In PBF-LB/M simulations, it is common to discretize the spatial part using the finite element method and combine it with a finite-difference-based

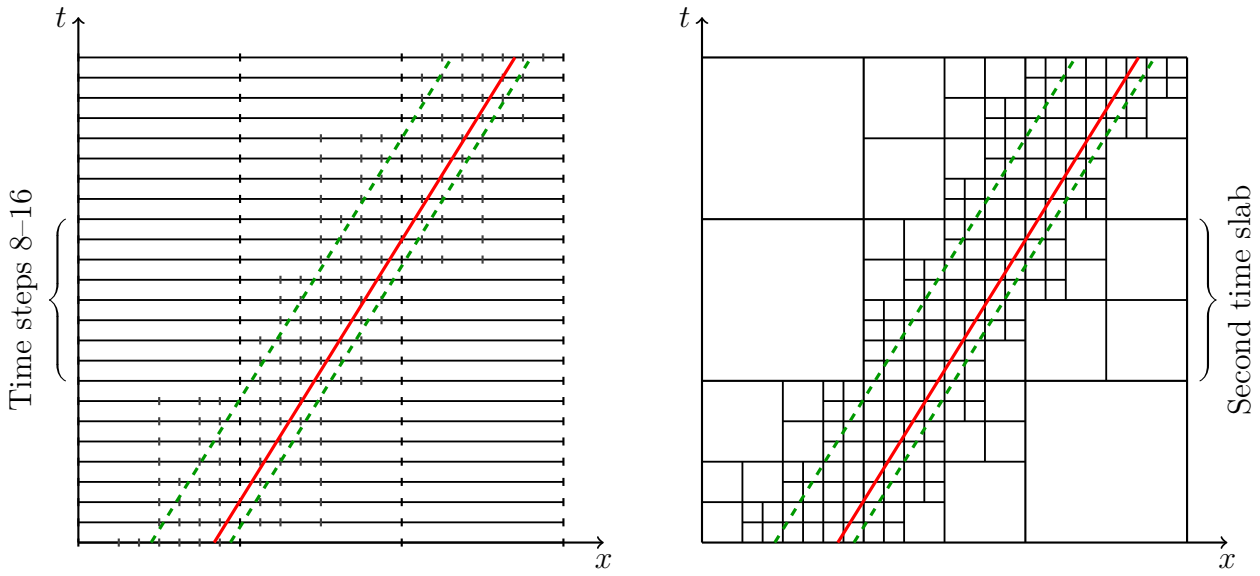
time-stepping scheme. This allows a local refinement of the finite element mesh during each time step towards the current position of the laser. The theoretical background of finite elements and their combination with time-stepping schemes can be found in Hughes (2000). The fundamental challenge in simulating PBF-LB/M processes is that detailed mathematical models are computationally expensive, and short fractions of the process already require significant resources on current supercomputers. Conversely, models that can be solved for the entire manufacturing process may not capture enough physical effects to gain the desired insight into the process. A review of current simulation models for metal AM can be found in Bayat et al. (2021).

To reliably predict the melt pool dynamics and related effects, such as lack of fusion or keyhole porosity, the fluid flow in the melt pool and the influence of evaporation must be included in the model. Resolving these effects on a scale of a few hundred micrometers over a time span of minutes or hours far exceeds the capabilities of any current simulation framework. However, the thermal evolution can already deliver valuable insights as most phenomena in PBF are driven by heat transfer and distribution. For example, thermal models can predict melt pool shapes (Kollmannsberger et al., 2019; Paulson et al., 2020), detect local overheating zones (Ranjan et al., 2020), determine the microstructure formation (Gu et al., 2018; Nitzler et al., 2021), or characterize the mechanical properties of the printed structure (Xie et al., 2021). The simplicity of thermal models combined with a lower spatial and temporal resolution of the laser spot results in much faster computations compared to higher fidelity models.

Part-scale simulations based on thermal models with time-stepping discretizations are still very challenging and often impossible when fully resolving the laser path without using lumped heat sources (that impose the equivalent heat of an entire line or layer at once). Classical approaches discretize only the spatial dimensions with finite elements and use a finite-difference scheme to advance the temperature in time. In particular, *hp*-methods can discretize the individual time steps efficiently by locally refining the mesh towards the laser spot (*h*-refinement) while increasing the polynomial degree (*p*-refinement) in regions with a smooth temperature field (Kollmannsberger et al., 2017; Kollmannsberger & Kopp, 2021). Two major challenges exist for developing part-scale time-stepping methods for simulating PBF-LB/M processes. First, they are restricted to the same time step length everywhere in space, leading to needlessly high temporal accuracy in regions further away from the laser. Depending on how expensive these peripheral regions are to compute, the cost for one time step may become prohibitively high. Second, efficiently using high-performance computing (HPC) resources requires sufficiently large problem sizes. However, the simulation of significant parts of the printing process often requires millions of time steps, leaving much less than one second to compute individual time steps. Even if a capable adaptive spatial discretization reduces the computational time to a few seconds on a single CPU, then a single matrix-vector multiplication in the iterative solution of the linear equation system takes only a few milliseconds. For such small problems, the communication overhead makes obtaining good parallel scaling difficult.

1.3 Space-time *hp*-finite element discretizations

This thesis introduces an alternative way of simulating the thermal evolution in PBF-LB/M that uses a four-dimensional finite element interpolation for uniformly discretizing space and



(a) Time-stepping with local refinement in space (b) Locally refined space-time finite elements

Figure 1.7: Discretization method comparison for a transient problem in one spatial dimension with a moving heat source (source path in red, refinement region in dashed green). From Kopp, Calo, et al. (2022), used under Creative Commons CC BY 4.0 license.

time. In this setting, elements have three spatial lengths and a duration (the length in time). This allows refining the mesh in the combined four-dimensional space-time, such that finer elements around the laser spot span a shorter duration and coarser elements in the peripheral regions span a longer duration. The simulation duration is subdivided into a sequence of time slabs that are solved consecutively to maintain reasonable problem sizes. This provides the flexibility to increase the sizes of the linear systems originating from individual space-time slabs according to the target environment. While desktop computers benefit from smaller systems that are relatively cheap to compute, large HPC systems may require larger systems to reduce the overall communication overhead. Figures 1.7 and 1.8 demonstrate this idea in one and two space dimensions. Such a method requires a suitable space-time weak formulation that performs well for PBF-LB/M applications. As any PBF-LB/M process is three-dimensional in space, a capable method for constructing four-dimensional, locally refined *hp*-basis functions is necessary. To the author's knowledge, no space-time *hp*-finite element methods have been developed for simulating PBF-LB/M processes so far. However, several recent approaches (Soldner & Mergheim, 2019; Hodge, 2021; Cheng & Wagner, 2021; Viguerie et al., 2022) address the multi-scale nature of the solution by dividing the spatial domain into several subdomains that are integrated with different time step lengths. The continuity across the subdomains is enforced weakly, which allows to develop iterative schemes based on local solutions of the individual problems. Such strategies can reduce the computational effort of integrating peripheral regions in time, resulting in speedups over conventional methods with a uniform time step length. The continuous space-time approach presented herein is compatible in space and time by construction, and heterogeneous time step lengths (i.e., element durations) are an inherent property of the method.

The standard Bubnov-Galerkin finite element discretization for symmetric problems like

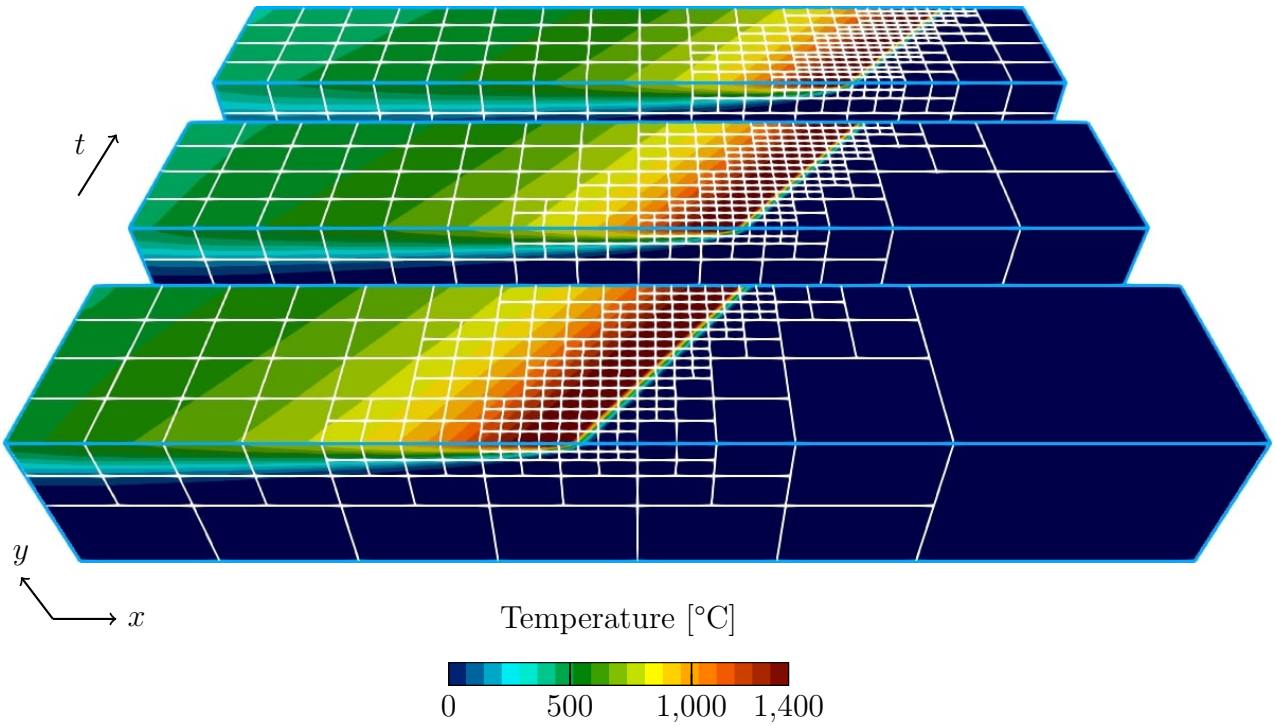


Figure 1.8: Three refined space-time slabs for a nonlinear heat equation in two spatial dimensions. From Kopp, Calo, et al. (2022), used under Creative Commons CC BY 4.0 license.

Poisson's equation uses the same space for the test and trial functions and results in the best approximation (Hughes, 2000). Unfortunately, this result does not apply to the standard space-time formulation of the heat equation. The first derivative in time leads to an unsymmetric bilinear form with different continuity requirements in the test and trial spaces, rendering it a formulation of Petrov-Galerkin type. The space-time bilinear form does not define an inner product, and the finite element solution is not a projection in a naturally induced (energy) norm. Showing uniform stability and quasi-optimality for pairs of discrete spaces for such Petrov-Galerkin formulations is challenging (Führer & Karkulik, 2021). Steinbach (2015) derives optimal convergence estimates of order p for standard continuous finite elements on general space-time meshes, where p is the polynomial degree of the finite elements. They require that the test space contains the trial space, and they later choose the same space for both. They measure the error in a norm that is H^1 in space and L^2 in time. Initial numerical experiments with the same C^0 test and trial spaces show suboptimal convergence rates in the space-time L^2-norm for even polynomial degrees and non-local spurious oscillations that travel backward in time in the presence of sharp, underresolved features in the solution. Moreover, a C^0 continuous test space does not directly allow the separation of the global space-time problem into consecutive time slabs, which requires a discontinuous test space across certain time slices.

An alternative approach based on a discontinuous test space was introduced by Aziz and Monk (1989), where the test functions on each temporal interval are polynomials of degree $p-1$ in time that are not connected across the temporal interfaces. This method leads to optimal convergence in the L^2-norm and yields block-triangular systems that can be solved consecutively. Under certain regularity conditions, the method is even superconvergent at the

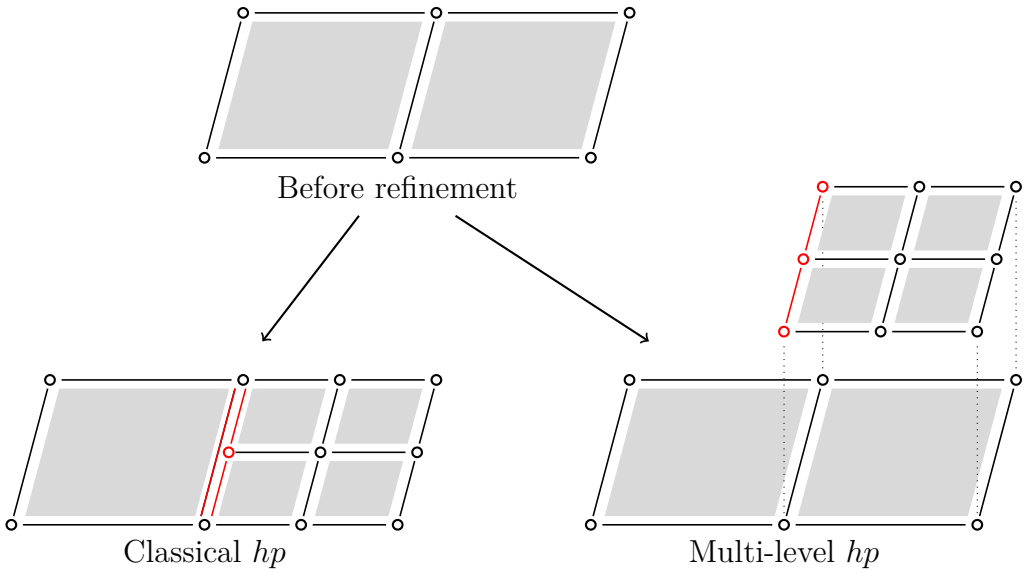


Figure 1.9: Comparison of refine-by-replacement and refine-by-superposition strategies.

temporal mesh points. The authors also present an equivalent formulation that tests with the time derivative of the trial functions (of degree p), which span the same space of piecewise discontinuous polynomials of degree $p-1$. They suggest that this approach can be used to construct adaptive space-time finite element methods, which is confirmed by the results of this thesis. Schieweck (2010) discusses an energy-decreasing property and combines Lagrange interpolations with a Gauss-Lobatto quadrature in time to decouple the linear systems, and Hussain et al. (2011) investigate suitable efficient parallel solution strategies. A similar space-time formulation for continuous finite element interpolations was developed by Devaud and Schwab (2018) and, in particular, Langer et al. (2019) and Langer and Yang (2020), who use hierarchical B-Splines to construct an adaptive isogeometric space-time method. Loli et al. (2022) add a high-order stabilization term to the isogeometric formulation to obtain a triangular system in time that can be advanced step-wise. Discontinuous Galerkin (DG) methods in time do not force continuity across element interfaces, even in the trial space, and are often used to develop time-stepping schemes that are adaptive in space (Jamet, 1978; Eriksson et al., 1985). Due to the reduced continuity, DG approaches lead to more unknowns, rendering them less efficient than continuous methods. An overview and comparison of space-time finite element methods for parabolic problems can be found in Akrivis et al. (2011) and Steinbach and Yang (2019). Adaptive versions of the continuous and discontinuous Galerkin methods in time that separately refine the spatial and temporal discretizations depending on the error in a chosen quantity of interest were developed by Meidner and Vexler (2007) and Schmich and Vexler (2008).

Finite element discretizations that can perform *hp*-refinements are particularly suitable to approximate solutions that are smooth in large parts of the computational domain but have local features that must be resolved. Classical replacement-based *hp*-methods in three dimensions refine by substituting several fine elements in place of an original coarse element (for example, eight fine cubes for one coarse cube). On the resulting incompatible interfaces between coarse and fine elements, called hanging nodes, the shape functions of finer elements must be constrained to the shape functions of the coarse neighbor. In higher dimensions, hanging nodes

also refer to incompatible edges, faces, and so on. These constraints already develop complex dependencies in three dimensions (Demkowicz et al., 1989; Demkowicz, 2006). The multi-level *hp*-method introduced in Zander et al. (2015), Zander et al. (2016), Di Stolfo et al. (2016), and Zander et al. (2022) simplifies the *hp*-basis construction by keeping the coarse elements in a hierarchical data structure and essentially constructing a high-order finite element basis on the meshes of each refinement level independently. Figure 1.9 demonstrates how hanging nodes (nodes and edges) appear when replacing coarse elements and shows the alternative superposition approach followed by the multi-level *hp*-method. There, the treatment of hanging nodes is transformed into a simple deactivation of overlay nodes on the left side. Further refinements are possible by adding more hierarchical overlays with finer elements. The method builds an object-oriented data structure to formulate two simple rules for obtaining linear independent and C^0 continuous basis functions that are complete up to a selected polynomial degree on each leaf cell of the refinement tree. While this object-oriented data structure works well in 1D, 2D, and 3D, the number of entities grows exponentially for higher dimensions. In 3D, one refined cube knows about its six faces, twelve edges, eight nodes, and eight child cubes (the fine elements); each face knows about its four edges, four nodes, and four child faces; each edge knows about the two nodes and two child edges; each vertex knows about its child vertex. Additionally, all topological components carry a list of connected elements (cubes) to compare their refinement levels and determine whether the component shall be active or inactive. In four dimensions, one hypercube comprises eight cubes, twenty-four faces, thirty-two edges, and sixteen nodes. Each again carries links to the topological sub-components, children, and adjacent elements. The complexity of constructing and maintaining such a data structure in four dimensions is significant, and efficient implementations with low memory management overhead and favorable memory access patterns become increasingly challenging to develop.

To address the complexity of *hp*-refinements in four dimensions, an even simpler data structure is introduced in this thesis that only stores the direct topological relations between the cells of the refinement tree (cubes in three dimensions or hypercubes in higher dimensions) and not their sub-components (faces, edges, nodes). A consequence of this simplification is that nodes, for example, are not explicitly stored anymore, and one cannot find all connecting elements across a corner of an element by simply reading the list of connected elements of that node. While not explicitly stored, this information still exists implicitly within the list of the neighbors across the 2^d interfaces for all elements, where d is the number of dimensions. Therefore, this reduced data structure is complemented by a set of algorithms that construct compatible *p*- and *hp*-basis functions equivalent to the original multi-level *hp*-method by implicitly extracting much of the topological information rather than relying on explicit storage. This is achieved by introducing a *tensor-product mask* and a *location matrix* for each cell of the refinement tree that determines the activation state and the global basis function index of its shape functions and formulating algorithms in terms of operations on the *array slices* of these containers. By repeating these algorithms several times, the information travels across multiple interfaces and reaches, for example, all elements connected to a node. Only the leaves of the refinement tree are considered finite elements, as they form a non-overlapping partition of the computational domain. This view hides the hierarchical nature to allow the use of standard finite element procedures without any modifications. The combination with the aforementioned continuous Petrov-Galerkin formulation leads to a stable and efficient space-time method for simulating PBF-LB/M processes.

1.4 Outline

This thesis is structured as follows. Chapter 2 discusses a thermal model with a volumetric heat source (Section 2.1) and an apparent heat capacity (Section 2.2) to consider the latent heat of fusion. The time-stepping method introduced in Section 2.3 uses finite elements to discretize in space and is the reference for evaluating the performance of the space-time finite element method. The presentation of the space-time finite element method is split into three parts, starting with introducing the weak form in the continuous setting in Section 2.4. Section 2.5 analyses two formulations with a continuous and a discontinuous test space and compares their performance using two linear benchmark problems. Section 2.6 then discusses the solution of the nonlinear equation system arising from a space-time finite element discretization of the nonlinear heat equation. Chapter 3 introduces the essential concepts for constructing p-finite element bases for an arbitrary number of dimensions, starting with the necessary topological information of a p-finite element mesh in Section 3.1. Section 3.2 discusses tensor-products of the integrated Legendre polynomials used as shape functions on each finite element and the construction of the associated tensor-product masks (Section 3.3) and location matrices (Section 3.4). These contain information on what shape functions are active and what global basis function index they contribute to. Section 3.5 introduces the construction of trunk space initial tensor-product masks that only keep the essential functions required for completeness. These concepts are then applied to multiple levels of p-finite elements in Chapter 4 to construct hp-bases defined on the hierarchical mesh discussed in Section 4.1. The major difference in the construction of tensor-product masks (Section 4.2) and location matrices (Section 4.3) compared to Chapter 3 is the distinction between internal interfaces and internal boundaries. These concepts are embedded into a standard finite element simulation workflow in Section 4.4. Sections 4.5 and 4.6 then introduce an a priori refinement strategy tailored to PBF-LB/M processes and the compatible separation of the four-dimensional hp-meshes into consecutive time slabs by introducing additional ghost slabs across the initial and final time slices of each slab. The presented approach is applied to several examples in Chapter 5. A verification in Section 5.1 shows that the multi-level hp extension behaves optimally for a simple Poisson problem. Section 5.2 then validates the presented space-time method on the AMB2018 benchmark case. The same setup is used in Section 5.3 to hatch a square centimeter for about one second and a path length of around one meter. As the last example, Section 5.4 compares the performance of the space-time and time-stepping approaches. Finally, Chapter 6 summarizes the main results of the thesis and discusses possible future extensions and applications.

The presented multi-level hp extension was pre-published in Kopp, Rank, et al. (2022). The content of Section 2.3, Chapter 3, Chapter 4 until after Section 4.4, and Section 5.1 is taken from this publication, including literal transcription. Similarly, the application to thermal space-time PBF-LB/M simulations was pre-published in Kopp, Calo, et al. (2022). The content of Chapter 2 without Section 2.3, Sections 4.5 – 4.6, and Sections 5.2 – 5.3 is taken from this publication, including literal transcription. Footnotes [1] and [2] reference the respective publications in the concerning sections and mark potential literal transcription.

[1] The following content is based on Kopp, Rank, et al. (2022). The main scientific research and its textual elaboration was performed by the author of this work.

[2] The following content is based on Kopp, Calo, et al. (2022). The main scientific research and its textual elaboration was performed by the author of this work.

Chapter 2

Formulation[1,2]

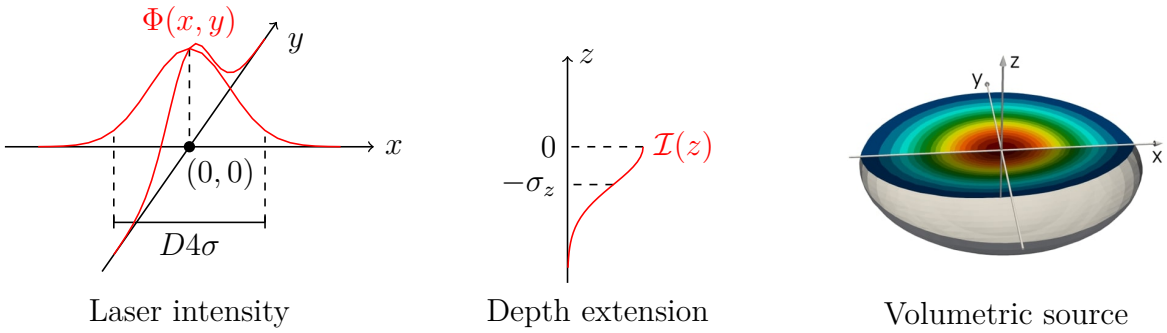
This chapter discusses the thermal model and introduces its time-stepping and Petrov-Galerkin space-time finite element formulations. The strong form of the nonlinear heat equation is defined as follows:

$$\begin{aligned} c\dot{u} + \nabla \cdot (k\nabla u) &= f && \text{on } \Omega = \mathcal{S} \times \mathcal{T} \\ u &= u_0 && \text{on } \mathcal{S}, \text{ at } t = t_0 \\ u &= g && \text{on } \Gamma_D \\ n \cdot k\nabla u &= h && \text{on } \Gamma_N. \end{aligned} \tag{2.1}$$

The space-time domain Ω is the product of a d-dimensional spatial domain $\mathcal{S}$ (considered time-invariant in this thesis) and a time interval $\mathcal{T} = (t_0, t_1]$; u is the temperature, $\dot{u}$ and ∇u are the time derivative and spatial gradient of the temperature. Moreover, $c = c(u)$ and $k = k(u)$ are the heat capacity and the heat conductivity of the material, f is the volumetric source function, g and h are the prescribed temperature and heat flux on the Neumann and Dirichlet boundaries Γ_N and Γ_D, and u_0 is the initial condition. The Neumann and Dirichlet boundaries do not intersect ($\Gamma_D \cap \Gamma_N = \emptyset$), and they together form the spatial boundary during the simulation ($\Gamma_D \cup \Gamma_N = \partial\mathcal{S} \times \mathcal{T}$). The heat flux $c(u)$ is split into the density and specific heat capacity $c(u) = \rho c_s(u)$. In practice, ρ also depends on the temperature, but it is assumed constant for simplicity.

The following sections use the terms time slab, time slice, and time step to refer to parts of the space-time domain Ω. A *time slab* extracts a piece of Ω on a temporal subinterval $(t_i, t_{i+1}] \in \mathcal{T}$ and is defined as $\mathcal{S} \times (t_i, t_{i+1}]$. Time slabs have the same $d + 1$ dimensions as Ω and often refer to the subdomains on which individual space-time finite element problems are computed. In contrast, *time slices* are d-dimensional spatial "cuts" through Ω at times t_i, defined as $\mathcal{S} \times t_i$. In specific contexts, a time slice or time slab may refer to the temperature field on the respective subdomain instead of the subdomain itself. A *time step* is used in the context of time-stepping methods and consists of a pair of time values (t_i, t_{i+1}) with a duration or time step length of $\Delta t_i = t_{i+1} - t_i$. The time-stepping scheme discussed in Section 2.3 formulates a finite element problem for advancing a known (spatial) temperature field on time slice t_i to the new temperature field on time slice t_{i+1}. While time-stepping schemes operate on temperature slices at discrete times, space-time formulations (like the one introduced in Section 2.4) interpolate the temperature field on space-time slabs.

2.1 Laser model[2]



Laser intensity Depth extension Volumetric source

Figure 2.1: Volumetric extension of the two-dimensional laser intensity profile in z-direction.

The laser heat input model has a significant impact on the quality of the thermal solution. Most current PBF-LB/M systems use lasers with wavelengths in the infrared spectrum, spatially distributed over a Gaussian profile. The laser intensity is therefore modeled as

$$\Phi(x, y) = \frac{P}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right),$$

where P is the total power of the laser and σ is the width of the laser beam shape (often given as $D4\sigma = 4\sigma$). The coordinate system is chosen so that the build plate and subsequent layers lie in the $x - y$ plane, and z is the vertical direction parallel to the laser beam. Other shapes are possible and currently being investigated for improving the process quality and productivity, for example, in Grünewald et al. (2021). In addition to the laser shape, the laser path $p : t \to (x, y, z)$ is given after slicing the geometric model and scanning the interior areas.

When encountering a metal surface, the electromagnetic waves are partially reflected and absorbed, depending on the absorptivity of the material. With higher absorptivities, more radiant energy of the electromagnetic waves is absorbed and converted into thermal energy on the surface of the material. The absorptivity changes with factors such as temperature, surface roughness, and angle of incidence, but these effects are often condensed into a single absorptivity ν. For low temperatures and flat substrate surfaces, the heat input can be imposed as a distributed heat flux on the top surface with

$$h(x, y) = \nu\, \Phi(x - p_x(t), y - p_y(t)). \tag{2.2}$$

Such surface models deliver increasingly poorer results after the laser power is high enough to reach melting and, eventually, boiling temperatures below the laser spot. Once evaporation becomes dominant and recoil pressure-induced depressions and keyholes form, the effective absorptivities dramatically increase, as shown by Trapp et al. (2017), for example. Besides absorbing more energy, the heat enters below the original top surface, which has been melted and pushed into the substrate by the recoil pressure. This increases the depth of the melt pool more than a surface heat source model can predict, even when adjusting the effective absorptivity accordingly.

Capturing these effects requires a detailed model that includes fluid flow and evaporation. However, choosing a volumetric heat input can improve the results from thermal models if

there is not enough recoil pressure for keyholing. By allowing heat to be introduced over some depth in the substrate, the melt pool depth can be calibrated to a given setup. To this end, a penetration intensity function $\mathcal{I}(z)$ is selected, for example

$$\mathcal{I}(z) = \frac{1}{\sqrt{2\pi}\sigma_z} \exp\left(-\frac{z^2}{2\sigma_z^2}\right), \tag{2.3}$$

where σ_z is the penetration depth that can be chosen in relation to the laser parameters. Then, the volumetric source function f is defined as

$$f(x, y, z, t) = \nu\, \mathcal{I}(z - p_z(t))\, \Phi(x - p_x(t), y - p_y(t)). \tag{2.4}$$

Figure 2.1 sketches the combination of Φ and $\mathcal{I}$ into a volumetric heat source. Allowing the laser to penetrate the substrate to some degree is also beneficial when using a continuum approach to model metal powder. On this scale, resolving individual grains is not feasible, but the natural penetration of the laser light into the powder is still important. As Gusarov et al. (2009) suggest, a volumetric heat source with an extension into the material similar to (2.3) can account for this effect in a homogenized way. Similar models have been analyzed by Zhang et al. (2018), Kollmannsberger et al. (2019), and Imani Shahabad et al. (2020).

2.2 Phase change model[2]

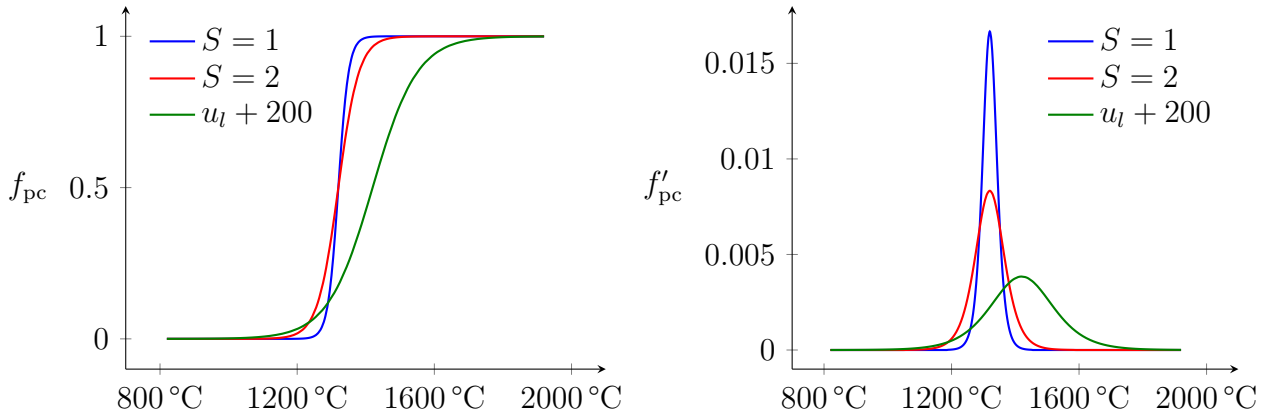
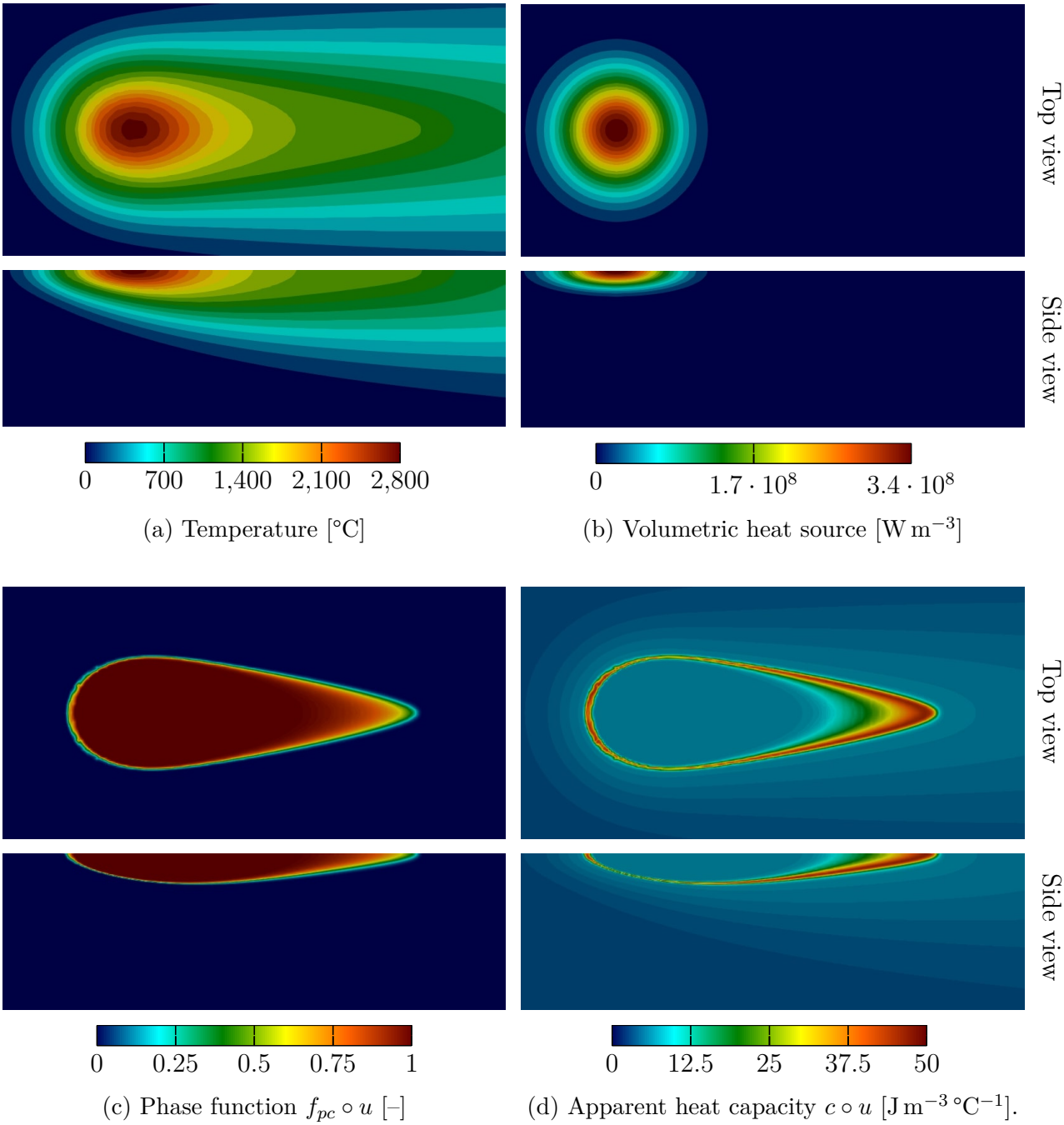


Figure 2.2: Phase change regularization between solid and liquid states. From Kopp, Calo, et al. (2022), used under Creative Commons CC BY 4.0 license.

The heat capacity of a material specifies how much energy is required to increase its temperature by a fixed amount. The same energy is released when the material cools down to the original temperature. This becomes quite extreme during phase transitions, where a significant amount of latent heat must be spent in completing the phase transition before the temperature can rise again. For simulating PBF-LB/M processes, the latent heat of fusion is essential for obtaining more realistic temperatures in the melt pool. The otherwise higher temperatures and temperature gradients lead to an artificially high diffusion of heat that results in overestimating the melt pool width and depth and underestimating the melt pool length. While the



(a) Temperature [°C]

(b) Volumetric heat source [$\mathrm{W\,m^{-3}}$]

(c) Phase function $f_{pc} \circ u$ [–]

(d) Apparent heat capacity $c \circ u$ [$\mathrm{J\,m^{-3}\,°C^{-1}}$].

Figure 2.3: Phase change between $u_s = 1290\,°\mathrm{C}$ and $u_l = 1350\,°\mathrm{C}$ with $S = 1$. The laser travels from right to left on top of an IN625 metal plate (see Section 5.2).

melting of pure metals is isothermal, the alloys commonly used in PBF-LB/M usually melt over a temperature range that depends on the composition.

There are many ways of including the latent heat of fusion into a finite element simulation. In particular, single-field approaches that do not introduce two separate solution fields for the solid and liquid phases are attractive as they are comparably simple to implement. The primary solution can either be the temperature or the enthalpy, from which the temperature can be recovered afterwards. Temperature formulations are attractive for non-isothermal

phase changes where the transition can be described continuously. To this end, a phase change function $f_{\mathrm{pc}}(u)$ is defined, with a value of zero for fully solidified material and one for fully liquid material. The interpolation between the solid temperature u_s and the liquid temperature u_l is typically done using a continuous function, for example,

$$f_{\mathrm{pc}}(u) = \frac{1}{2}\left(\tanh\left(\frac{u-u_m}{u_\sigma}\right)+1\right), \tag{2.5}$$

where u_m and u_σ are the melting temperature and the width of the phase change, defined as

$$u_m = \frac{u_l+u_s}{2} \qquad u_\sigma = S\,\frac{u_l-u_s}{2}. \tag{2.6}$$

The regularization parameter S gives additional control over the smoothness of the phase change, which is later used to improve the numerical behavior of the problem. The formulation of (2.5) smoothly transitions between the solid and liquid states, containing most of the latent heat of fusion between u_s and u_l. The first and second derivatives of f_{pc} with respect to temperature are:

$$f'_{\mathrm{pc}}(u) = \frac{1}{2u_\sigma}\left(1-\tanh\left(\frac{u-u_m}{u_\sigma}\right)^2\right)$$

$$f''_{\mathrm{pc}}(u) = \frac{1}{u_\sigma^2}\tanh\left(\frac{u-u_m}{u_\sigma}\right)\left(\tanh\left(\frac{u-u_m}{u_\sigma}\right)^2-1\right).$$

Figure 2.3a shows a temperature field computed from the volumetric source in Figure 2.3b, using the material and laser parameters of Section 5.2. The composition $u \circ f_{pc}$ of the temperature u and the phase change function f_{pc} evaluates to one inside the melt pool and transitions to zero elsewhere in a very narrow zone, as Figure 2.3c shows.

Together with the latent heat of fusion L of the material, the heat equation (2.1) is augmented with a latent heat term that stores the energy of the phase change, and hence its time derivative acts as a heat source or sink:

$$\rho c_s \dot{u} + \rho L \dot{f}_{\mathrm{pc}} + \nabla\cdot(k\nabla u) = f \tag{2.7}$$

By applying the chain rule

$$\dot{f_{\mathrm{pc}}} = \frac{\partial f_{\mathrm{pc}}(u)}{\partial t} = \frac{\mathrm{d}f_{\mathrm{pc}}}{\mathrm{d}u}\frac{\partial u}{\partial t} = f'_{\mathrm{pc}}\dot{u},$$

the standard form of the heat equation (2.1) can be recovered:

$$\left(\rho c_s + \rho L f'_{\mathrm{pc}}\right)\dot{u} + \nabla\cdot(k\nabla u) = f. \tag{2.8}$$

In (2.8), $c(u) = \rho c_s(u) + \rho L f'_{\mathrm{pc}}(u)$ is called the apparent heat capacity. This formulation is particularly attractive as it can be used with existing heat equation solvers and only requires modifying the heat capacity. The derivative of f_{pc} adds a sharp spike around the melting temperature, as Figure 2.3d shows. This makes the numerical quadrature of the finite element integrals challenging, especially since thermal models for PBF-LB/M often use only few elements to discretize the melt pool volume. Figure 2.3, however, uses $39 \times 13 \times 7$ third-order

finite elements across a melt pool with a size of about 400 µm × 130 µm × 35 µm. Reducing the number of elements in this discretization requires further regularization of f_{pc} (by increasing S) to smooth the spike in c.

The formulation of (2.7) allows time-stepping schemes to introduce a separate finite difference approximation for $\dot{f}_{pc}$. Such schemes offer improved stability over formulations based on (2.8), as discussed in Section 2.3. Technically, (2.7) can be used for isothermal phase transitions, but in practice, it still may be necessary to regularize the discontinuous phase change function f_{pc}. Because (2.7) does not naturally transfer to a single field space-time formulation, Section 2.4 uses the apparent heat capacity version (2.8) instead.

2.3 Time-stepping finite element formulation[1]

This section discusses a time-stepping finite element formulation of (2.7) with dynamic refinement and derefinement to compare to the space-time formulation introduced in Section 2.4, which is the main focus of this thesis. The derivation follows closely the work presented by Celentano et al. (1994) but uses Rothe's method, where time is discretized first, instead of using the method of lines, where space is discretized first. The formulation in terms of spatial functions on the sequence of time slices allows the subsequent spatial discretization to change between time steps, which is crucial for using dynamic *hp*-refinement with a time-stepping scheme. Such approaches are commonly used in adaptive finite elements for time-dependent problems (Kollmannsberger & Kopp, 2021; Kopp, Rank, et al., 2022). This thesis uses the Wilson-θ scheme to discretize in time, which results in the conditionally stable, explicit, first-order accurate Forward-Euler scheme for $\theta = 0$, the unconditionally stable, implicit, second-order accurate Crank-Nickolson scheme for $\theta = 1/2$, and the unconditionally stable, implicit, first-order accurate Backward-Euler scheme for $\theta = 1$. Importantly, these characteristics hold only for linear problems under certain regularity assumptions.

A time-stepping scheme subdivides the time interval $\mathcal{T}$ into a sequence of time steps from t^n to t^{n+1} and sequentially advances the solution on the associated time slices. Within a time step, the time derivatives are approximated as

$$\rho c_s \frac{\partial u}{\partial t} \approx \rho c_s^{n+\theta} \frac{u^{n+1} - u^n}{\Delta t} \qquad \text{and} \qquad \rho L \frac{\partial f_{\mathrm{pc}}}{\partial t} \approx \rho L \frac{f_{\mathrm{pc}}^{n+1} - f_{\mathrm{pc}}^n}{\Delta t},$$

where the n and $n+1$ superscripts represent time slices at t^n and t^{n+1}, and the superscript $n+\theta$ indicates interpolation at θ between the values at t^n and t^{n+1}:

$$c^{n+\theta} = (1-\theta)\, c^n + \theta\, c^{n+1}.$$

The remaining terms of (2.7) without a time derivative are interpolated at $t^{n+\theta}$ to obtain the following time discretization:

$$\rho c_s^{n+\theta} \frac{u^{n+1} - u^n}{\Delta t} + \rho L \frac{f_{\mathrm{pc}}^{n+1} - f_{\mathrm{pc}}^n}{\Delta t} - \nabla \cdot \Big[k\, \nabla u\Big]^{n+\theta} = f^{n+\theta}, \tag{2.9}$$

where

$$\Big[k\, \nabla u\Big]^{n+\theta} = (1-\theta)\, k(u^n)\, \nabla u^n + \theta\, k(u^{n+1})\, \nabla u^{n+1},$$

$$f^{n+\theta} = (1-\theta)\, f^n + \theta f^{n+1}.$$

The weak form of (2.9) then reads: Find $u^{n+1} \in u_g^{n+1} + H_0^1(\mathcal{S})$ such that

$$\int_{\mathcal{S}} \frac{\rho c_s^{n+\theta}}{\Delta t}\, w\, \left(u^{n+1} - u^n\right) + \frac{\rho L}{\Delta t}\, w\, \left(f_{\mathrm{pc}}^{n+1} - f_{\mathrm{pc}}^n\right) + \nabla w \cdot \left[k\, \nabla u\right]^{n+\theta} \mathrm{d}\mathcal{S} = \int_{\mathcal{S}} w f^{n+\theta}\, \mathrm{d}\mathcal{S} + \int_{\mathcal{N}} w h^{n+\theta}\, \mathrm{d}\mathcal{N}, \quad (2.10)$$

holds for all $w \in H_0^1(\mathcal{S})$. In (2.10), $\mathcal{N}$ refers to the spatial component of the Neumann boundary Γ_N, such that $\mathcal{N} \times \mathcal{T} = \Gamma_N$, and $u_g^{n+1} \in H^1(\mathcal{S})$ is a smooth extension of the Dirichlet function g into the domain interior. The Sobolev space $H_0^1(\mathcal{S})$ contains real-valued functions defined on $\mathcal{S}$ with zero values on the Dirichlet boundary and weakly defined first derivatives. Sobolev spaces state the necessary continuity conditions and are essential in the mathematical analysis of finite elements. Selecting a finite element subspace $W^{n,h} \subset H_0^1$ for each time slice yields the interpolations $u_h^n = \sum N_i^n \hat{u}_i^n \in W^{n,h}$, where $\hat{u}_i^n$ are the space- and time-independent coefficients (degrees of freedom) for time slice n. As the time slices are discretized independently, the finite element meshes may differ, enabling the use of dynamic mesh refinement. In one time step, the basis $\{N_i^n\}$ for the current time slice and the basis $\{N_i^{n+1}\}$ for the next time slice do not have to coincide. Similarly, the number of coefficients may differ between the discretizations of different time slices. From now on, the superscript in N_i^{n+1} is omitted to simplify the notation. Using again the same test as trial functions allows to obtain the discrete weak residual:

$$R_i^n(\hat{u}^{n+1}) = \int_{\mathcal{S}} \frac{\rho c_s^{n+\theta}}{\Delta t}\, N_i\, \left(u_h^{n+1} - u_h^n\right) + \frac{\rho L}{\Delta t}\, N_i\, \left(f_{\mathrm{pc}}^{n+1} - f_{\mathrm{pc}}^n\right) + \nabla N_i \cdot \left[k\, \nabla u_h\right]^{n+\theta} \mathrm{d}\mathcal{S} - \int_{\mathcal{S}} N_i f^{n+\theta}\, \mathrm{d}\mathcal{S} - \int_{\mathcal{N}} N_i h^{n+\theta}\, \mathrm{d}\mathcal{N}. \quad (2.11)$$

The material parameters c_s and k and the phase change function f_{pc} are now evaluated using the discrete solution. Again the $n+\theta$ superscripts indicate an interpolation at θ between the evaluations at t^n and t^{n+1}. Constructing an iterative scheme with second-order convergence to find the coefficients $\hat{u}^{n+1}$ that solve $R_i^n(\hat{u}^{n+1}) = 0$ requires deriving the linearization $\partial R_i^n / \partial \hat{u}_j^{n+1}$. Using the intermediate results

$$\frac{\partial u_h^{n+1}}{\partial \hat{u}_j^{n+1}} = N_j, \qquad \frac{\partial u_h^n}{\partial \hat{u}_j^{n+1}} = 0,$$

and, therefore,

$$\frac{\partial c_s^{n+\theta}}{\partial \hat{u}_j^{n+1}} = (1-\theta)\, c_s'^n \frac{\partial u_h^n}{\partial \hat{u}_j^{n+1}} + \theta\, c_s'^{n+1} \frac{\partial u_h^{n+1}}{\partial \hat{u}_j^{n+1}} = \theta\, c_s'^{n+1} N_j,$$

$$\frac{\partial}{\partial \hat{u}_j^{n+1}} \left[c_s^{n+\theta}\, \left(u_h^{n+1} - u_h^n\right)\right] = c_s^{n+\theta} \frac{\partial u_h^{n+1}}{\partial \hat{u}_j^{n+1}} + \frac{\partial c_s^{n+\theta}}{\partial \hat{u}_j^{n+1}} \left(u_h^{n+1} - u_h^n\right)$$

$$= c_s^{n+\theta}\, N_j + \theta\, c_s'^{n+1} \left(u_h^{n+1} - u_h^n\right) N_j$$

$$
\begin{aligned}
\frac{\partial}{\partial \hat{u}_j^{n+1}}\Big[f_{\mathrm{pc}}^{n+1} - f_{\mathrm{pc}}^{n}\Big] &= f_{\mathrm{pc}}'^{\,n+1}\,\frac{\partial u_h^{n+1}}{\partial \hat{u}_j^{n+1}} = f_{\mathrm{pc}}'^{\,n+1} N_j \\
\frac{\partial}{\partial \hat{u}_j^{n+1}}\Big[k\,\nabla u_h\Big]^{n+\theta} &= \theta k^{n+1}\frac{\partial\left(\nabla u_h^{n+1}\right)}{\partial \hat{u}_j^{n+1}} + \theta\, k'^{\,n+1}\,\frac{\partial u_h^{n+1}}{\partial \hat{u}_j^{n+1}}\nabla u_h^{n+1} \\
&= \theta k^{n+1}\nabla N_j + \theta\, k'^{\,n+1}\, N_j \nabla u_h^{n+1},
\end{aligned}
$$

the consistent linearization of (2.11) becomes:

$$
\begin{aligned}
T_{ij}^n(\hat{u}^{n+1}) = &\int_{\mathcal{S}} \frac{\rho}{\Delta t}\left(c_s^{n+\theta} + \theta\, c_s'^{\,n+1}\Big(u_h^{n+1} - u_h^n\Big) + L f_{\mathrm{pc}}'^{\,n+1}\right) N_i N_j \,\mathrm{d}\mathcal{S} \\
&+ \int_{\mathcal{S}} \theta\,\nabla N_i \cdot \left(k^{n+1}\nabla N_j + k'^{\,n+1}\nabla u_h^{n+1} N_j\right)\,\mathrm{d}\mathcal{S}.
\end{aligned}
\tag{2.12}
$$

The second "stiffness" term in (2.12) involving the derivative of the conductivity k' can be omitted to maintain symmetry of the linear systems. This is often done and does not reduce the convergence of the nonlinear iterations for the problems considered here by much; however, for highly nonlinear conductivities, it may be needed. Now, a nonlinear iteration scheme is constructed using (2.11) and (2.12), starting with an initial guess $\hat{u}^{n+1,\,0}$ for the coefficients of time $n+1$:

$$
\begin{aligned}
\hat{u}^{n+1,\,k+1} &= \hat{u}^{n+1,\,k} + \Delta\hat{u}^{n+1,\,k} \\
T_{ij}^n(\hat{u}^{n+1,\,k})\,\Delta\hat{u}_j^{n+1,\,k} &= -R_i^n(\hat{u}^{n+1,\,k}).
\end{aligned}
$$

The computations in this thesis use the symmetric linearization outlined in this section. Moreover, the L^2 projection of u_h^n onto the basis of time $n+1$ is used as an initial guess for the nonlinear iterations. This projection could also be used in the finite element integrals (2.12) and (2.11) instead of evaluating u_h^n using the old finite element basis. However, such a linear L^2 projection does not conserve energy if the heat capacity depends on temperature.

The representation of the solution in two time slices requires two finite element bases (for t^n and for t^{n+1}) to assemble the equation system resulting from one time step. The two bases are constructed on different meshes when dynamically refining and derefining, resulting in element interfaces located inside an element of the next mesh. The Gauss-Legendre quadrature used to integrate the finite element integrals is designed to integrate polynomials exactly but delivers poor results when integrating functions with low regularity. To prevent integrating over the C^0 continuous element interfaces of the previous mesh, an integration cell may not overlap more than one element per time slice mesh. The hierarchical refinements built in Chapter 4 allow local solution features to be resolved using the same base mesh in all time steps to capture the coarse scales, guaranteeing that finer elements of one mesh are always fully contained in the coarser element of the other mesh. Using this structure, the integration cells to assemble an element of the t^{n+1} mesh are found by first identifying the corresponding cell in the t^n mesh. This corresponding t^n cell either is either coarser, identical, or overlayed by several finer cells. The identical case is trivial and does not require special treatment. If the t^n cell is coarser, the quadrature points are distributed on the t^{n+1} element and mapped to the t^n cell. If the t^n cell is a coarse parent of finer elements, the quadrature points are distributed on the finer elements and mapped to the coarse parent (corresponding to the t^{n+1} element). The hierarchical mesh and its data structure are discussed in Section 4.1.

2.4 Continuous space-time weak formulation[2]

Instead of advancing the solution slice-by-slice, space-time finite elements discretize the entire space-time domain with a $d+1$ dimensional finite element interpolation. Multiplying (2.1) by a test function w and integrating by parts gives the following space-time weak form:
Find $u \in u_b + U_0$, such that

$$\int_\Omega w\, c\dot{u} + \nabla w \cdot k\nabla u \,\mathrm{d}\Omega = \int_\Omega w f \,\mathrm{d}\Omega + \int_{\Gamma_N} w\, h \,\mathrm{d}\Gamma_N \qquad \forall w \in W_0, \tag{2.13}$$

where $u_b \in U$ satisfies the initial and Dirichlet conditions. The test and trial spaces U and W incorporate the conditions on the functions w and u that allow the terms in (2.13) to be integrated in space and time. To integrate the second term, the spatial gradients of w and u must be defined (weakly), and their dot product must be integrable over space and time. While this condition is sufficient for w, the time derivative in the first term leads to the additional condition that $\dot{u}$ must be defined (at least in an inner product with w). Because the test and trial spaces differ, the method is of Petrov-Galerkin type.

The mathematical analysis of space-time finite elements is often based on Bochner spaces of functions from the time interval $\mathcal{T}$ into a spatial Sobolev space $H^s(\mathcal{S})$. In particular, the test and trial spaces

$$\begin{aligned} W &= L^2(\mathcal{T}; H^1(\mathcal{S})) \\ U &= L^2(\mathcal{T}; H^1(\mathcal{S})) \cap H^1(\mathcal{T}; H^{-1}(\mathcal{S})) \end{aligned}$$

define the existence conditions for (2.13). The test space W contains functions that for any time $t \in \mathcal{T}$ yield a spatial function in $H^1(\mathcal{S})$, such that its spatial energy norm (H^1-norm on $\mathcal{S}$) is square integrable in time (L^2-norm on $\mathcal{T}$). The trial space additionally requires that spatial slices of the time derivative are from H^{-1} and that their norm is square integrable in time. In other words, the second condition requires $\dot{u}$ to be defined in space, at least in the sense of distributions. It may contain a Dirac delta, for example, since the test functions w in the product $w\dot{u}$ are continuous in space, as they are from $H^1(\mathcal{S})$, and the product with an element from the dual space $H^{-1}(\mathcal{S})$ is well-defined in an integral sense. Moreover, the restrictions of W and U to functions that are zero on the Dirichlet boundary are again denoted by W_0 and U_0.

A weak form is well-posed if it has a unique solution that can be bounded by the source term (also called stability property; see Ern & Guermond, 2004, introduction of Chapter 2). The weak form of linear problems can be written as $b(u, w) = l(w)$, where b is called a bilinear form and l a linear functional. For elliptic problems, the well-posedness of a weak form follows from the Lax-Milgram theorem that requires $b(u, w)$ to be bounded and coercive (positive-definite). Unfortunately, Lax-Milgram cannot be used here, as (2.13) is parabolic, and the test and trial spaces in the bilinear form are different; instead, an inf-sup stability condition must be proven. In classic linear elasticity for structural mechanics, such inf-sup conditions can be interpreted as requiring that for every non-zero displacement field (trial function), there is at least one admissible virtual displacement (test function) that causes a non-zero virtual work (the result of evaluating the bilinear form). Of course, this context does not apply to the space-time formulation of the heat equation, but inf-sup conditions do not need such engineering interpretations. They are the continuous equivalent of requiring the (stiffness) matrix to be

invertible in the discrete setting (see Ern & Guermond, 2004, Remark 2.23). Schwab and Stevenson (2009) show that (2.13) is well-posed, permitting a unique solution u. Interestingly, stationary advection-diffusion problems have a very similar structure with a first and second derivative. However, the standard weak form is coercive as the diffusion acts in all directions.

2.5 Space-time finite element discretization[2]

A finite element approximation of (2.13) constructs discrete subspaces $U^h \subset U$ and $W^h \subset W$ on a finite element mesh. In contrast to the discretization of the time slices in Section 2.3, the mesh and the basis functions are now $d+1$ dimensional as they include the temporal dimension. Unfortunately, the additional complexity related to the continuous parabolic problem transfers to the discrete setting. For elliptic problems, the well-posedness result of the Lax-Milgram theorem transfers to the discrete setting (Ern & Guermond, 2004, Proposition 2.19), and the natural choice $W^h = U^h$ even yields the best approximation for symmetric bilinear forms (Hughes, 2000, Chapter 4.1). For parabolic problems, stability must be proven again for each pair of discrete spaces U^h and W^h, and there is no obvious choice, such as selecting $W^h = U^h$ for elliptic problems.

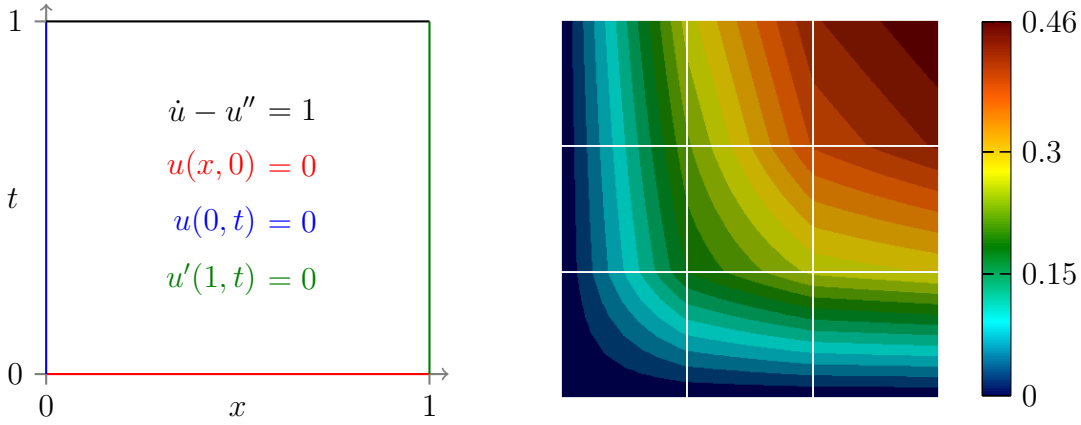
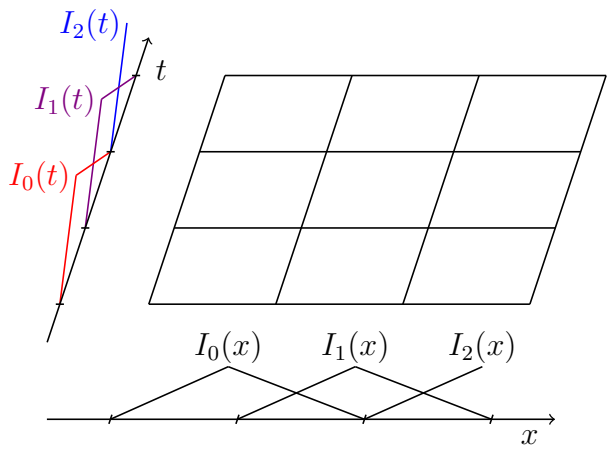
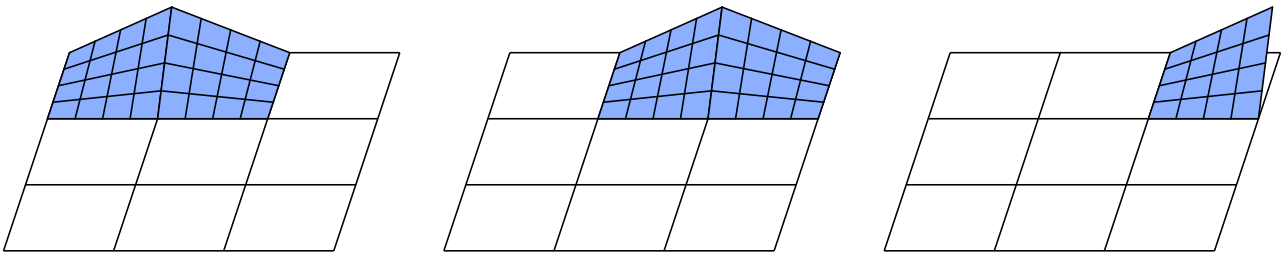


Figure 2.4: Problem setup for comparing the matrix structure of three test bases and their numerical solution on a 3×3 mesh.

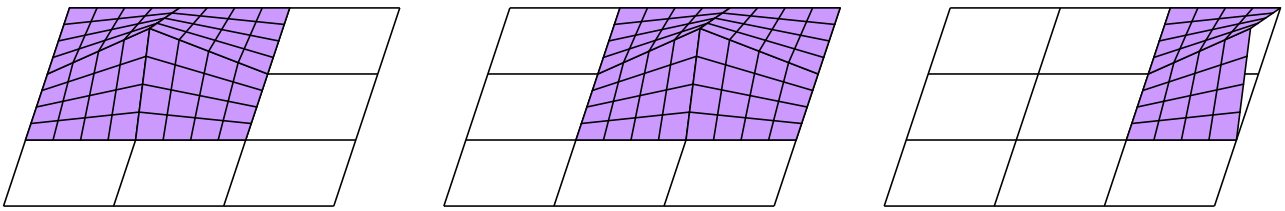
This section compares two space-time finite element methods that use standard continuous piecewise polynomials to represent the solution, but their test spaces are different. Figure 2.4 shows a simple linear heat problem with one spatial and one temporal dimension. Its discretization in Figure 2.5 uses the space-time tensor-product of three piecewise linear "hat" functions in space and time, resulting in nine basis functions $N_i(x,t)$. This thesis uses such standard finite element basis functions to span the trial spaces in space and time, although local refinements and high-order basis functions are added later to improve the approximation properties. The first method uses the same test and trial functions, which is intuitive and easy to implement in existing finite element codes. Steinbach (2015) derives the discrete stability of this method in a mesh-dependent norm and shows first and second-order convergence for $p = 1$ and $p = 2$ in the energy norm (H^1 norm in space, L^2 norm in time). However, unlike in symmetric, second-order problems, continuity in time is not necessary for the test functions. The second method tests with piecewise constant functions supported only on one element



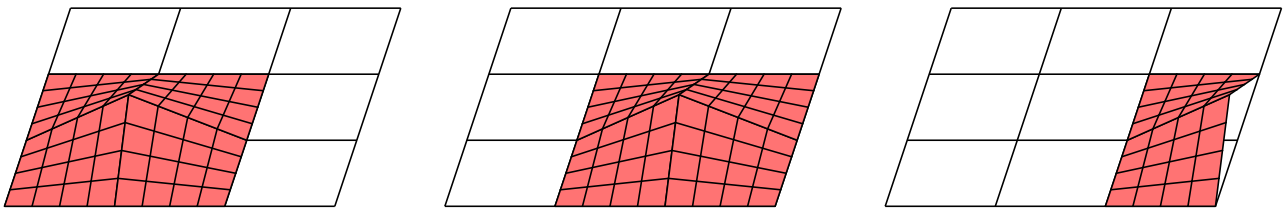
$$\boldsymbol{N}(x,t) = \begin{pmatrix} N_0(x,t) \\ N_1(x,t) \\ N_2(x,t) \\ N_3(x,t) \\ N_4(x,t) \\ N_5(x,t) \\ N_6(x,t) \\ N_7(x,t) \\ N_8(x,t) \end{pmatrix} = \begin{pmatrix} I_0(x) \cdot I_0(t) \\ I_1(x) \cdot I_0(t) \\ I_2(x) \cdot I_0(t) \\ I_0(x) \cdot I_1(t) \\ I_1(x) \cdot I_1(t) \\ I_2(x) \cdot I_1(t) \\ I_0(x) \cdot I_2(t) \\ I_1(x) \cdot I_2(t) \\ I_2(x) \cdot I_2(t) \end{pmatrix}$$



$N_6(x,t) = I_0(x) \cdot I_2(t)$ $N_7(x,t) = I_1(x) \cdot I_2(t)$ $N_8(x,t) = I_2(x) \cdot I_2(t)$

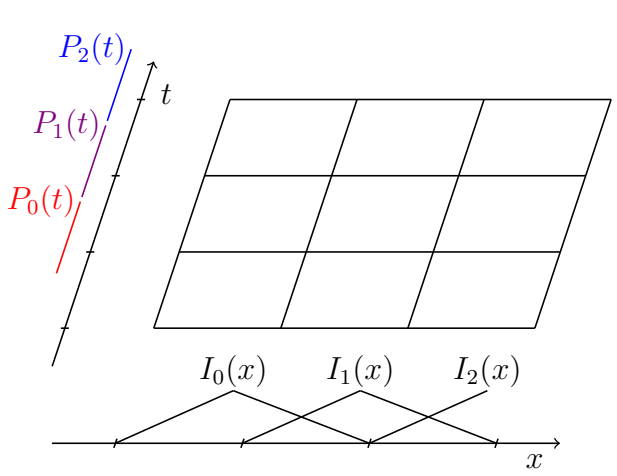


$N_3(x,t) = I_0(x) \cdot I_1(t)$ $N_4(x,t) = I_1(x) \cdot I_1(t)$ $N_5(x,t) = I_2(x) \cdot I_1(t)$

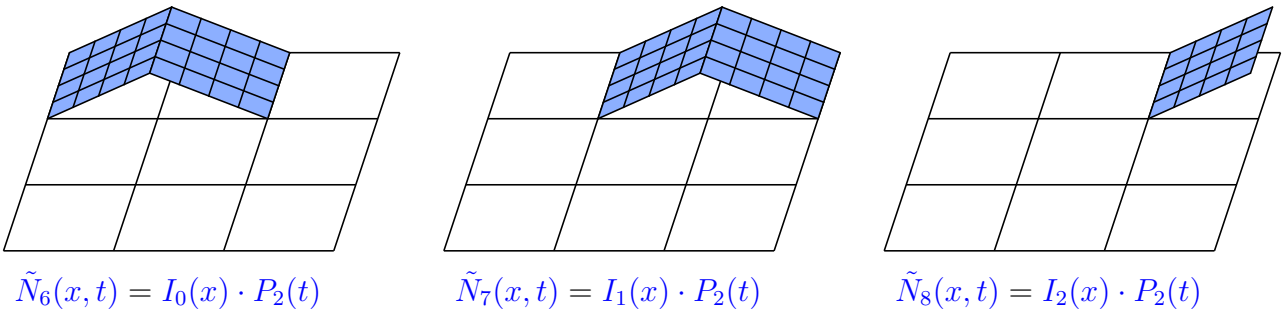


$N_0(x,t) = I_0(x) \cdot I_0(t)$ $N_1(x,t) = I_1(x) \cdot I_0(t)$ $N_2(x,t) = I_2(x) \cdot I_0(t)$

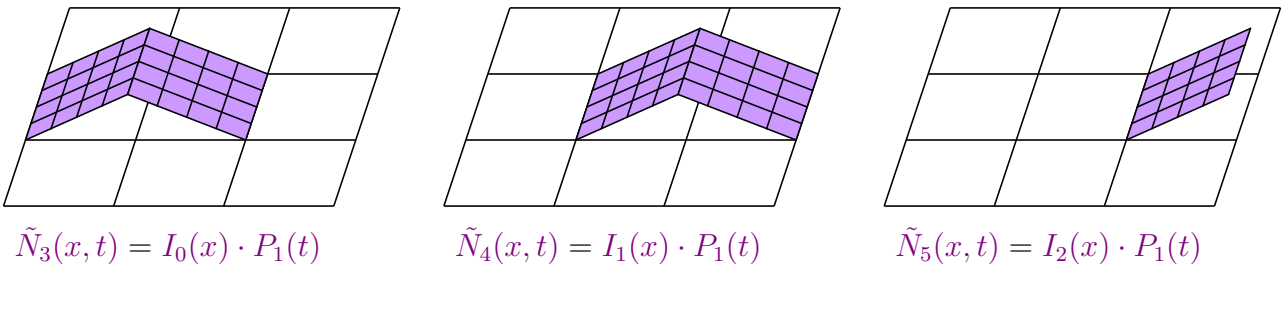
Figure 2.5: Space-time C^0 basis functions from the tensor-product of piecewise linear "hat" functions after imposing a zero initial condition on the bottom edge and Dirichlet condition on the left edge.



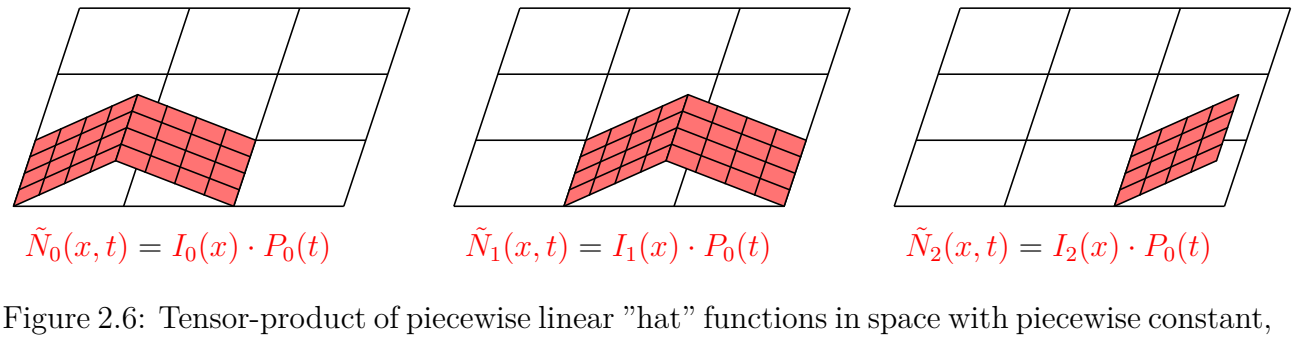
$$\boldsymbol{N}(x,t) = \begin{pmatrix} \tilde{N}_0(x,t) \\ \tilde{N}_1(x,t) \\ \tilde{N}_2(x,t) \\ \tilde{N}_3(x,t) \\ \tilde{N}_4(x,t) \\ \tilde{N}_5(x,t) \\ \tilde{N}_6(x,t) \\ \tilde{N}_7(x,t) \\ \tilde{N}_8(x,t) \end{pmatrix} = \begin{pmatrix} I_0(x) \cdot P_0(t) \\ I_1(x) \cdot P_0(t) \\ I_2(x) \cdot P_0(t) \\ I_0(x) \cdot P_1(t) \\ I_1(x) \cdot P_1(t) \\ I_2(x) \cdot P_1(t) \\ I_0(x) \cdot P_2(t) \\ I_1(x) \cdot P_2(t) \\ I_2(x) \cdot P_2(t) \end{pmatrix}$$



$\tilde{N}_6(x,t) = I_0(x) \cdot P_2(t)$ $\tilde{N}_7(x,t) = I_1(x) \cdot P_2(t)$ $\tilde{N}_8(x,t) = I_2(x) \cdot P_2(t)$

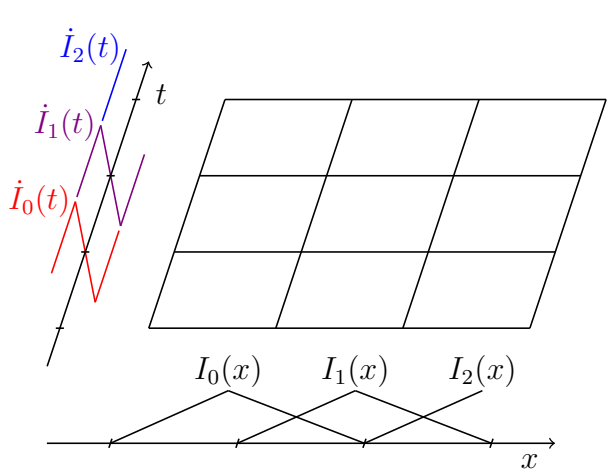


$\tilde{N}_3(x,t) = I_0(x) \cdot P_1(t)$ $\tilde{N}_4(x,t) = I_1(x) \cdot P_1(t)$ $\tilde{N}_5(x,t) = I_2(x) \cdot P_1(t)$

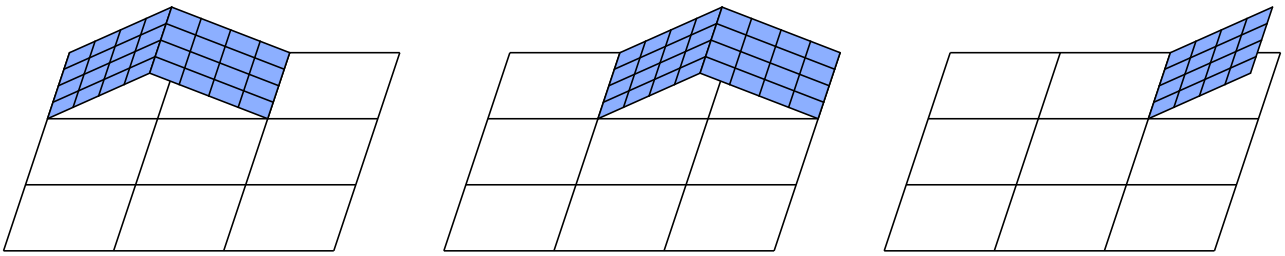


$\tilde{N}_0(x,t) = I_0(x) \cdot P_0(t)$ $\tilde{N}_1(x,t) = I_1(x) \cdot P_0(t)$ $\tilde{N}_2(x,t) = I_2(x) \cdot P_0(t)$

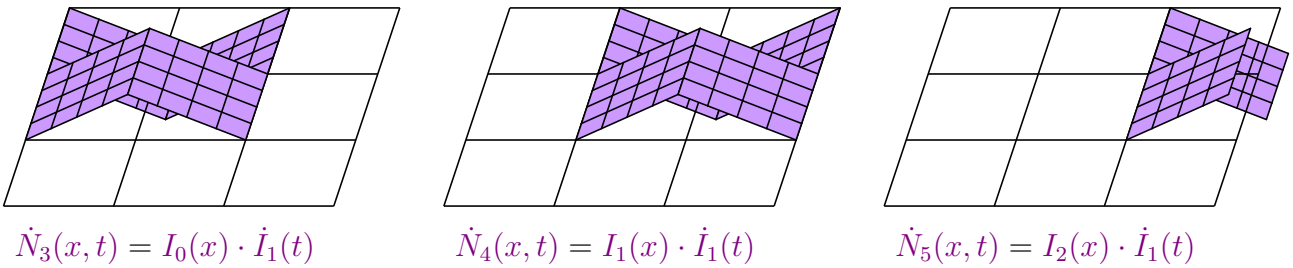
Figure 2.6: Tensor-product of piecewise linear "hat" functions in space with piecewise constant, discontinuous functions in time.



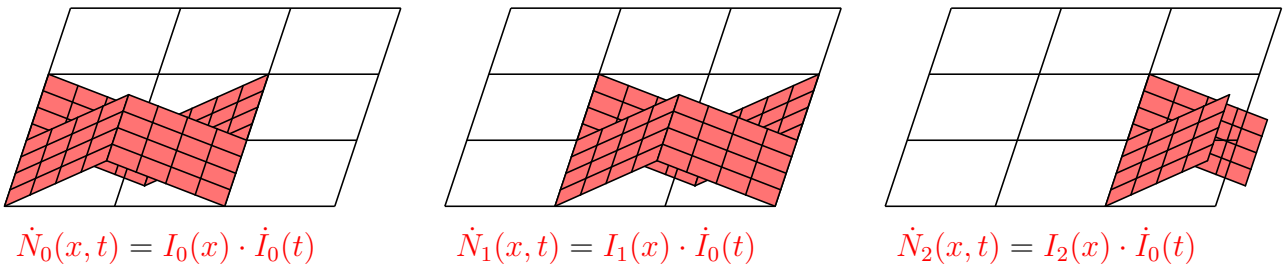
$$\boldsymbol{N}(x,t) = \begin{pmatrix} \dot{N}_0(x,t) \\ \dot{N}_1(x,t) \\ \dot{N}_2(x,t) \\ \dot{N}_3(x,t) \\ \dot{N}_4(x,t) \\ \dot{N}_5(x,t) \\ \dot{N}_6(x,t) \\ \dot{N}_7(x,t) \\ \dot{N}_8(x,t) \end{pmatrix} = \begin{pmatrix} I_0(x) \cdot \dot{I}_0(t) \\ I_1(x) \cdot \dot{I}_0(t) \\ I_2(x) \cdot \dot{I}_0(t) \\ I_0(x) \cdot \dot{I}_1(t) \\ I_1(x) \cdot \dot{I}_1(t) \\ I_2(x) \cdot \dot{I}_1(t) \\ I_0(x) \cdot \dot{I}_2(t) \\ I_1(x) \cdot \dot{I}_2(t) \\ I_2(x) \cdot \dot{I}_2(t) \end{pmatrix}$$



$\dot{N}_6(x,t) = I_0(x) \cdot \dot{I}_2(t)$ $\dot{N}_7(x,t) = I_1(x) \cdot \dot{I}_2(t)$ $\dot{N}_8(x,t) = I_2(x) \cdot \dot{I}_2(t)$

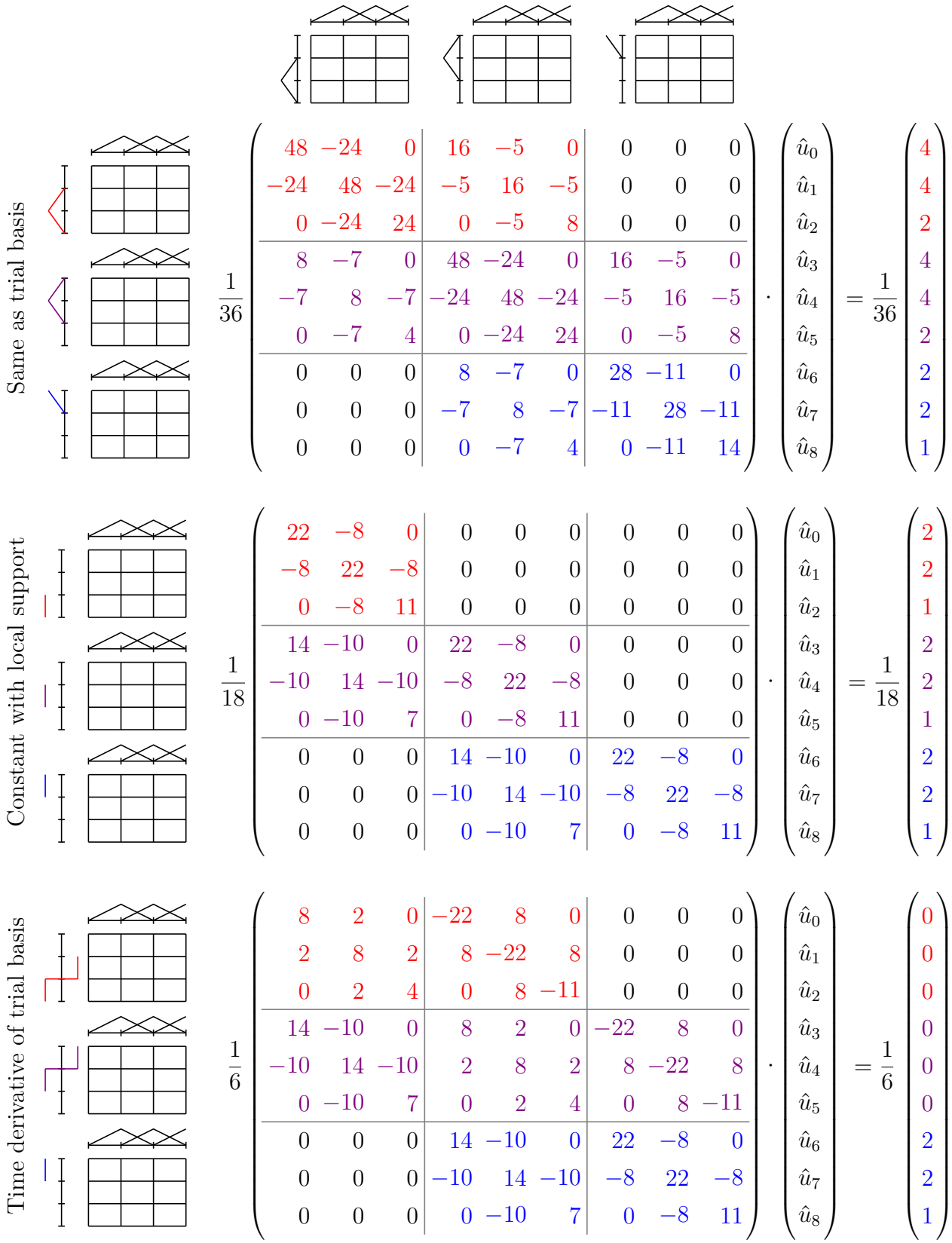


$\dot{N}_3(x,t) = I_0(x) \cdot \dot{I}_1(t)$ $\dot{N}_4(x,t) = I_1(x) \cdot \dot{I}_1(t)$ $\dot{N}_5(x,t) = I_2(x) \cdot \dot{I}_1(t)$



$\dot{N}_0(x,t) = I_0(x) \cdot \dot{I}_0(t)$ $\dot{N}_1(x,t) = I_1(x) \cdot \dot{I}_0(t)$ $\dot{N}_2(x,t) = I_2(x) \cdot \dot{I}_0(t)$

Figure 2.7: Time derivatives of the standard C^0 basis functions from Figure 2.5, spanning the same space as the basis in Figure 2.6.



$$\text{Same as trial basis:}\quad \frac{1}{36}\begin{pmatrix} 48 & -24 & 0 & 16 & -5 & 0 & 0 & 0 & 0 \\ -24 & 48 & -24 & -5 & 16 & -5 & 0 & 0 & 0 \\ 0 & -24 & 24 & 0 & -5 & 8 & 0 & 0 & 0 \\ \hline 8 & -7 & 0 & 48 & -24 & 0 & 16 & -5 & 0 \\ -7 & 8 & -7 & -24 & 48 & -24 & -5 & 16 & -5 \\ 0 & -7 & 4 & 0 & -24 & 24 & 0 & -5 & 8 \\ \hline 0 & 0 & 0 & 8 & -7 & 0 & 28 & -11 & 0 \\ 0 & 0 & 0 & -7 & 8 & -7 & -11 & 28 & -11 \\ 0 & 0 & 0 & 0 & -7 & 4 & 0 & -11 & 14 \end{pmatrix} \cdot \begin{pmatrix} \hat{u}_0 \\ \hat{u}_1 \\ \hat{u}_2 \\ \hat{u}_3 \\ \hat{u}_4 \\ \hat{u}_5 \\ \hat{u}_6 \\ \hat{u}_7 \\ \hat{u}_8 \end{pmatrix} = \frac{1}{36}\begin{pmatrix} 4 \\ 4 \\ 2 \\ 4 \\ 4 \\ 2 \\ 2 \\ 2 \\ 1 \end{pmatrix}$$

$$\text{Constant with local support:}\quad \frac{1}{18}\begin{pmatrix} 22 & -8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -8 & 22 & -8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -8 & 11 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 14 & -10 & 0 & 22 & -8 & 0 & 0 & 0 & 0 \\ -10 & 14 & -10 & -8 & 22 & -8 & 0 & 0 & 0 \\ 0 & -10 & 7 & 0 & -8 & 11 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 14 & -10 & 0 & 22 & -8 & 0 \\ 0 & 0 & 0 & -10 & 14 & -10 & -8 & 22 & -8 \\ 0 & 0 & 0 & 0 & -10 & 7 & 0 & -8 & 11 \end{pmatrix} \cdot \begin{pmatrix} \hat{u}_0 \\ \hat{u}_1 \\ \hat{u}_2 \\ \hat{u}_3 \\ \hat{u}_4 \\ \hat{u}_5 \\ \hat{u}_6 \\ \hat{u}_7 \\ \hat{u}_8 \end{pmatrix} = \frac{1}{18}\begin{pmatrix} 2 \\ 2 \\ 1 \\ 2 \\ 2 \\ 1 \\ 2 \\ 2 \\ 1 \end{pmatrix}$$

$$\text{Time derivative of trial basis:}\quad \frac{1}{6}\begin{pmatrix} 8 & 2 & 0 & -22 & 8 & 0 & 0 & 0 & 0 \\ 2 & 8 & 2 & 8 & -22 & 8 & 0 & 0 & 0 \\ 0 & 2 & 4 & 0 & 8 & -11 & 0 & 0 & 0 \\ \hline 14 & -10 & 0 & 8 & 2 & 0 & -22 & 8 & 0 \\ -10 & 14 & -10 & 2 & 8 & 2 & 8 & -22 & 8 \\ 0 & -10 & 7 & 0 & 2 & 4 & 0 & 8 & -11 \\ \hline 0 & 0 & 0 & 14 & -10 & 0 & 22 & -8 & 0 \\ 0 & 0 & 0 & -10 & 14 & -10 & -8 & 22 & -8 \\ 0 & 0 & 0 & 0 & -10 & 7 & 0 & -8 & 11 \end{pmatrix} \cdot \begin{pmatrix} \hat{u}_0 \\ \hat{u}_1 \\ \hat{u}_2 \\ \hat{u}_3 \\ \hat{u}_4 \\ \hat{u}_5 \\ \hat{u}_6 \\ \hat{u}_7 \\ \hat{u}_8 \end{pmatrix} = \frac{1}{6}\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 2 \\ 2 \\ 1 \end{pmatrix}$$

Figure 2.8: Linear system comparison for zero temperatures on the initial and left boundaries.

instead of connecting pairs of linear shape functions in time to continuous basis functions with support on two elements. This approach was introduced by Aziz and Monk (1989) and is later referred to as the continuous Galerkin-Petrov method, see Schieweck (2010), Hussain et al. (2011). Figure 2.6 shows the resulting test basis that is still continuous in space but discontinuous in time. Interestingly, this approach reduces to a Crank-Nicolson time-stepping scheme when using the tensor-product of a spatial and a temporal mesh together with a trapezoidal quadrature in time (Aziz & Monk, 1989). The same piecewise discontinuous test space can also be obtained by using the time derivatives of the trial basis as a test basis. While the support of the test functions is larger in this case, both bases can represent the same piecewise constant functions in time, as Figure 2.7 shows. The advantage of this choice is the close relation to the trial functions, which facilitates an implementation into a standard finite element framework and conceptually simplifies the construction of *hp*-test bases.

For high-order finite element interpolations (trial bases), the first approach uses the same (high-order piecewise polynomial) basis functions to test as the trial basis. This thesis uses integrated Legendre polynomials to construct high-order finite elements, as Chapter 3 discusses. The second approach can be extended to higher-order bases by using Legendre polynomials (or any other polynomial basis). Since, conceptually, the test basis seeks to approximate a time derivative of the trial basis, it is discontinuous across the element interfaces in time. It is one degree lower than the trial basis, which is continuous across the element interfaces in time. Therefore, the test functions are again piecewise constants for a linear finite element interpolation. A second-order interpolation yields a constant and a linear test function in time on each element, and so on. Interestingly, this approach reduces the polynomial degree in time of the second term of the weak form in (2.13) to $p \cdot (p-1)$ instead of p^2 for continuous trial and test bases, where p is the polynomial degree of the temporal finite element interpolation. This allows using only p instead of $p+1$ Gauss-Legendre quadrature points in time while still integrating the finite element system exactly (for a time-independent material). As an alternative to using a local polynomial test basis of degree $p-1$ on each element, one can again use the time derivative of the trial basis. The resulting functions have a larger support but span the same space.

A major difference between these approaches is how they split into sequential, slab-wise schemes. Figure 2.8 sketches the linear systems that arise from evaluating the weak form (2.13) with different test functions and the piecewise linear trial basis. The matrix and vector entries correspond to the left- and right-hand sides of (2.13). The first equation system shows how the continuous test basis introduces a coupling across the element interfaces in time, preventing a split into separate systems. In contrast, the piecewise constant test functions (second equation system) are supported on one element only and allow the problem to be solved in three time slabs, containing three elements each. Figure 2.9 shows the decoupled systems that use the solution on the final time slice of the previous slab as initial condition. Lastly, in the third equation system of Figure 2.8, the test functions are the time derivatives of the trial functions that again result in a coupling. Interestingly, the time derivative test functions combined with a constant heat source lead to a counter-intuitive right-hand side, where the only non-zero entries originate from the test functions with support on the final (temporal) boundary.

The piecewise continuous and discontinuous test bases also behave differently when compared on a linear heat equation in one spatial dimension. Figure 2.10a shows a simple benchmark problem setup, and Figure 2.10b shows the corresponding analytical solution. While

$$\begin{pmatrix} 22 & -8 & 0 \\ -8 & 22 & -8 \\ 0 & -8 & 11 \end{pmatrix} \cdot \begin{pmatrix} \hat{u}_0 \\ \hat{u}_1 \\ \hat{u}_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix} \qquad \rightarrow \begin{pmatrix} \hat{u}_0 \\ \hat{u}_1 \\ \hat{u}_2 \end{pmatrix} \approx \begin{pmatrix} 0.19 \\ 0.26 \\ 0.28 \end{pmatrix}$$

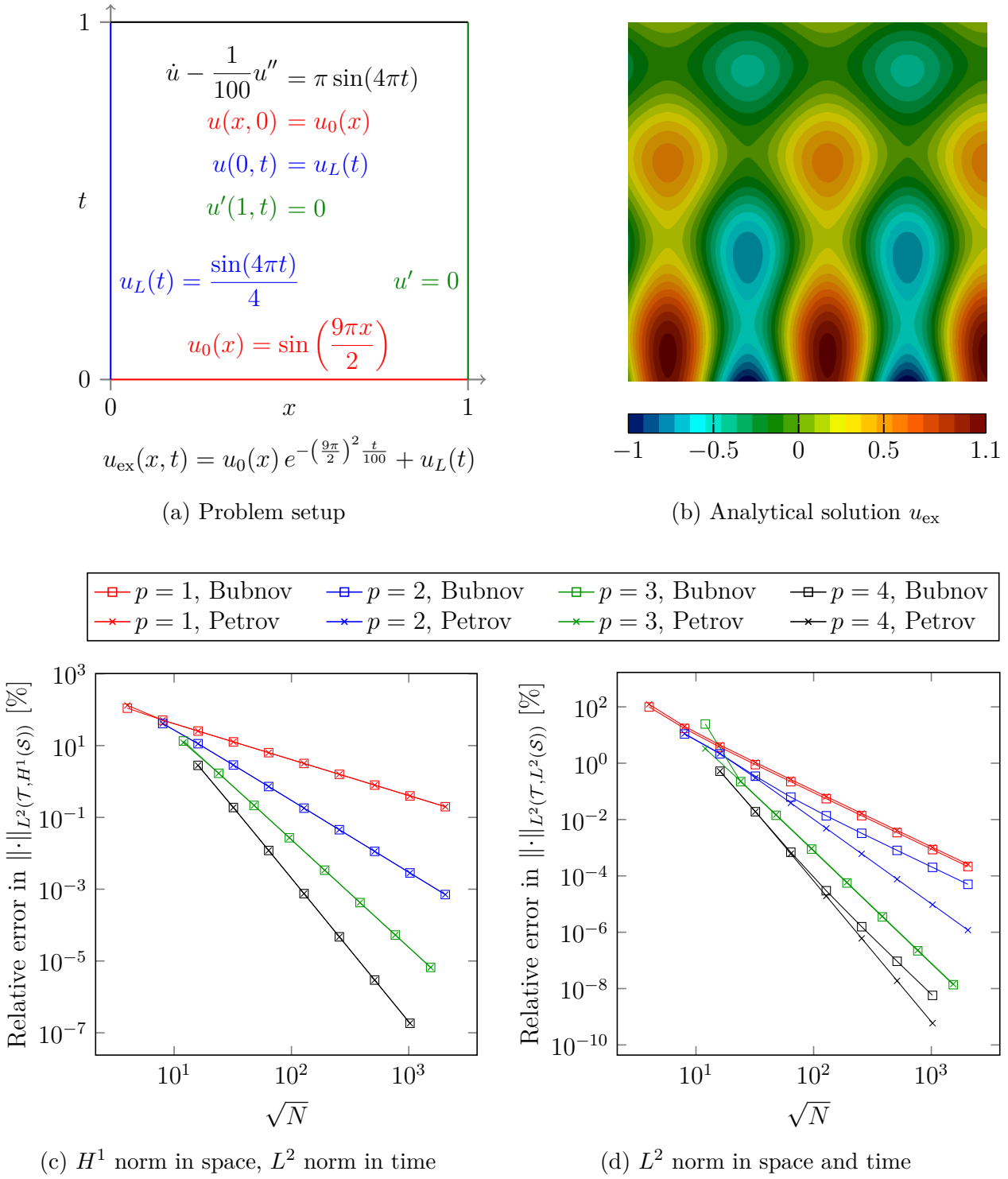
$$\begin{pmatrix} 22 & -8 & 0 \\ -8 & 22 & -8 \\ 0 & -8 & 11 \end{pmatrix} \cdot \begin{pmatrix} \hat{u}_3 \\ \hat{u}_4 \\ \hat{u}_5 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix} - \begin{pmatrix} 14 & -10 & 0 \\ -10 & 14 & -10 \\ 0 & -10 & 7 \end{pmatrix} \cdot \begin{pmatrix} 0.19 \\ 0.26 \\ 0.28 \end{pmatrix} \qquad \rightarrow \begin{pmatrix} \hat{u}_3 \\ \hat{u}_4 \\ \hat{u}_5 \end{pmatrix} \approx \begin{pmatrix} 0.23 \\ 0.37 \\ 0.42 \end{pmatrix}$$

$$\begin{pmatrix} 22 & -8 & 0 \\ -8 & 22 & -8 \\ 0 & -8 & 11 \end{pmatrix} \cdot \begin{pmatrix} \hat{u}_6 \\ \hat{u}_7 \\ \hat{u}_8 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix} - \begin{pmatrix} 14 & -10 & 0 \\ -10 & 14 & -10 \\ 0 & -10 & 7 \end{pmatrix} \cdot \begin{pmatrix} 0.23 \\ 0.37 \\ 0.42 \end{pmatrix} \qquad \rightarrow \begin{pmatrix} \hat{u}_6 \\ \hat{u}_7 \\ \hat{u}_8 \end{pmatrix} \approx \begin{pmatrix} 0.27 \\ 0.41 \\ 0.46 \end{pmatrix}$$

Figure 2.9: Sequential solution of the second system in Figure 2.8 after canceling 1/18 on both sides of the equation.

the errors of the two space-time methods converge equivalently in the H^1 norm in space (Figure 2.10c), the space-time L^2 convergence of the continuous test basis is one order lower for even polynomial degrees (Figure 2.10d). Figure 2.11 shows a similar setup with a narrow Gaussian source of varying widths in the center of the space-time domain. The solution interpolation uses 16×16 bilinear finite elements. The continuous test basis performs well initially (for $\sigma = 0.1$), but the solution quality deteriorates after decreasing the width of the point source. In particular, the discretization error results in spurious oscillations that are non-local and propagate backwards in time. In contrast, the discontinuous test basis performs much better. Although a discretization error is visible, its influence is local and does not violate causality. These observations do not depend on the quadrature order (this example uses $p + 10 = 11$ points per direction). Such sudden onsets or stops of the heat source are very common in PBF-LB/M, and suitable discretization methods must provide reasonable approximations even in the pre-asymptotic regime.

The properties outlined in this section motivate choosing a piecewise discontinuous basis with degree $p - 1$ for the space-time discretization of PBF-LB/M problems. The remaining task is to find an efficient implementation for this approach that can be used in a standard finite element framework. The time derivative approach is the most straightforward, but numerical experiments show that it may lead to an ill-conditioned global equation system. The element-local, piecewise discontinuous polynomial basis does not suffer from this issue, but the resulting element matrices are not square. Moreover, the shape functions must not be connected in time, which interferes with the multi-level *hp* rules introduced in Chapter 4. As a compromise, the time-derivative approach is modified by setting the rows of each element system to zero, that originate from the second shape function (with negative constant derivative). In the linear setting, this recovers the element-local piecewise constant approach: the negative contributions to the functions in Figure 2.7 are set to zero, resulting in the basis shown in Figure 2.6. The time-derivative of the additional high-order shape functions are element-local in time and do not cause any non-local interactions. Finally, the explicit zeros in the global equation system



(a) Problem setup

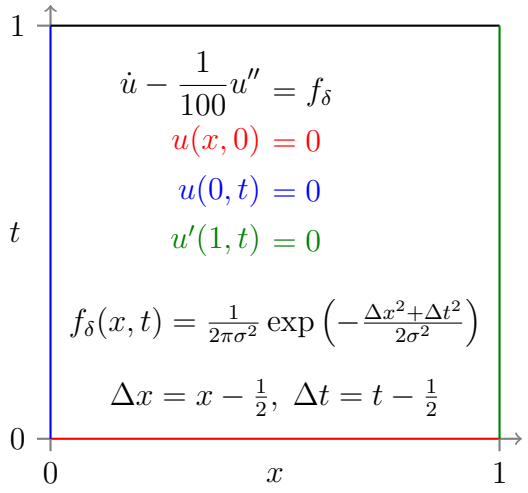
(b) Analytical solution u_{ex}

(c) H^1 norm in space, L^2 norm in time

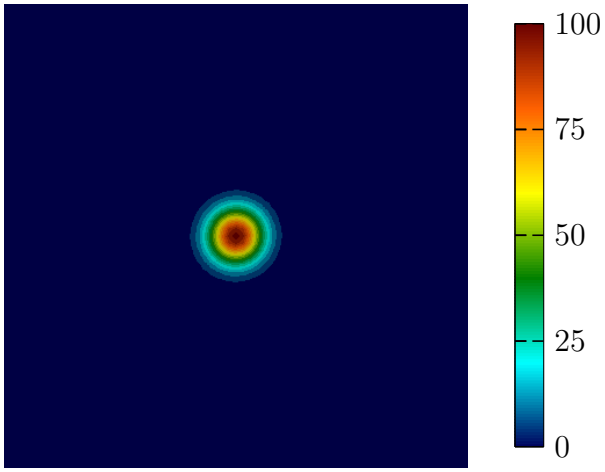
(d) L^2 norm in space and time

Figure 2.10: Comparison of the error convergence of the continuous and discontinuous test spaces for a linear heat equation in one (space) dimension with a smooth solution.

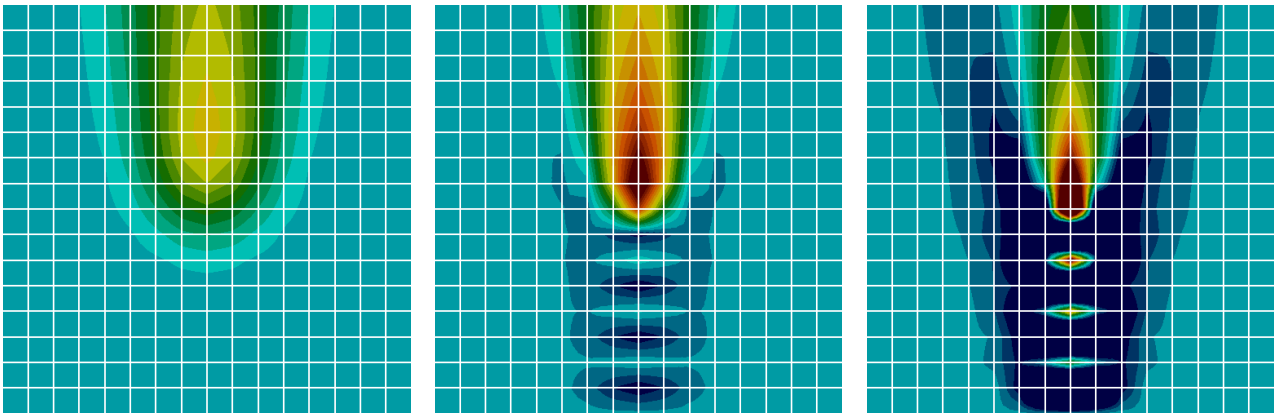
are filtered after finishing the assembly and before invoking the solution of the linear equation system.



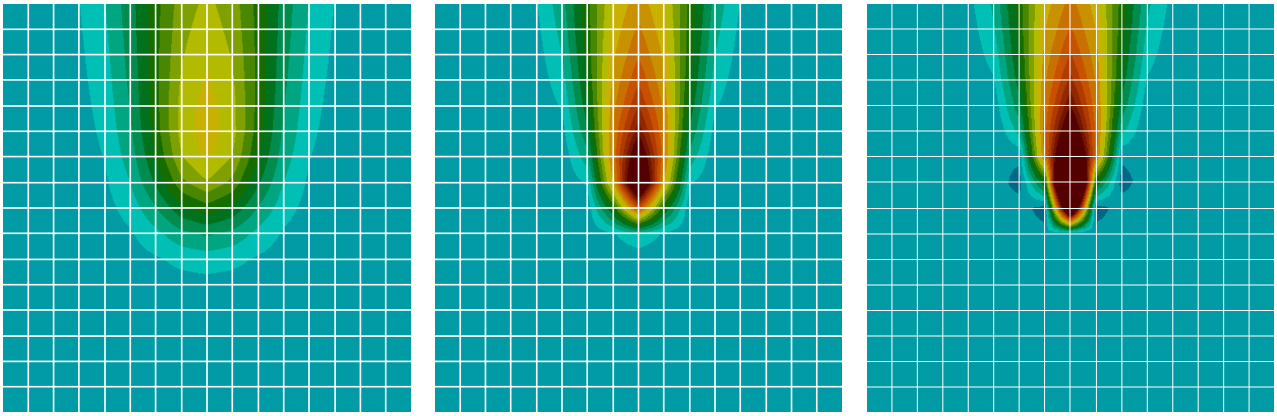
(a) Problem setup



(b) Heat source for $\sigma = 0.04$



(c) Continuous W^h, $\sigma = 0.1$ (d) Continuous W^h, $\sigma = 0.04$ (e) Continuous W^h, $\sigma = 0.01$



(f) Discontinuous W^h, $\sigma = 0.1$ (g) Discontinuous W^h, $\sigma = 0.04$ (h) Discontinuous W^h, $\sigma = 0.01$

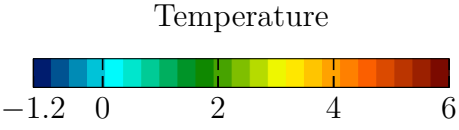


Figure 2.11: Causality violation of the continuous test space for an impulse heat source.

2.6 Solution of the nonlinear equation system[2]

The time derivative continuous Galerkin-Petrov method discussed in Section 2.5 results in the following discrete weak form:
Find $u^h \in u_b^h + W_0^h$, such that

$$\int_\Omega \dot{w}\, c\dot{u} + \nabla\dot{w} \cdot\, k\nabla u \,\mathrm{d}\Omega = \int_\Omega \dot{w} f \,\mathrm{d}\Omega + \int_{\Gamma_N} \dot{w}\, h \,\mathrm{d}\Gamma_N \qquad \forall w \in W_0(\Omega), \tag{2.14}$$

where W_0^h is a standard continuous finite element space (now including the temporal dimension) with the additional condition that the gradient of the time derivative is defined. This condition on the time derivative excludes elements such as triangles in space-time where the Jacobian matrix of the element mapping is not diagonal (block diagonal in higher dimensions). Still, the continuity of the temporal derivative condition allows for locally refined rectangular elements with hanging nodes, as shown in Figure 1.7, for example.

The temperature dependency of the coefficients c and k renders (2.14) nonlinear. The solution of the nonlinear finite element system is obtained using standard Newton-Raphson iterations

$$\hat{u}^{k+1} = \hat{u}^k + \Delta\hat{u}^k$$
$$T_{ij}(\hat{u}^k)\,\Delta\hat{u}_j^k = -R_i(\hat{u}^k),$$

around the temperature coefficients $\hat{u}^k$. The tangent matrix $T_{ij}(\hat{u}^k)$ and the residual $R_i(\hat{u}^k)$, evaluated at iteration k, are defined as follows:

$$R_i(u^h) = \int_\Omega \dot{N}_i\, c\dot{u}^h + \nabla\dot{N}_i \cdot\, k\nabla u^h - \dot{N}_i f \,\mathrm{d}\Omega - \int_{\Gamma_N} \dot{N}_i\, h \,\mathrm{d}\Gamma_N, \tag{2.15}$$

$$T_{ij}(u^h) = \frac{\partial R_i}{\partial \hat{u}_j} = \int_\Omega \dot{N}_i \left(c\dot{N}_j + c'\dot{u}^h N_j\right) + \nabla\dot{N}_i \cdot \left(k\nabla N_j + k'\nabla u^h N_j\right) \mathrm{d}\Omega, \tag{2.16}$$

where c' and k' are the derivatives of c and k with respect to temperature. The iterations start at $\hat{u}^0 = 0$ and stop once a reasonable reduction in the residual $\left\|R^k\right\| < \epsilon \left\|R^0\right\|$ is achieved. A value of $\epsilon = 10^{-4}$ was chosen for the computations in this thesis, as smaller values did not improve the solution noticeably.

The finite element systems are integrated in space with a standard Gauss-Legendre quadrature rule with $p+1$ points in each spatial direction. As the test functions have a maximum polynomial degree of $p-1$ in time, it may be sufficient to use only p quadrature points in time to integrate the products of shape functions. However, more quadrature points may be needed around a concentrated heat source as the rapid temperature changes and the material's nonlinearities can make the integrals quite rough. This quadrature inadequacy is especially problematic when using an apparent heat capacity model to account for the phase change, as Section 2.2 discussed.

Chapter 3

The *p*-finite element method[1]

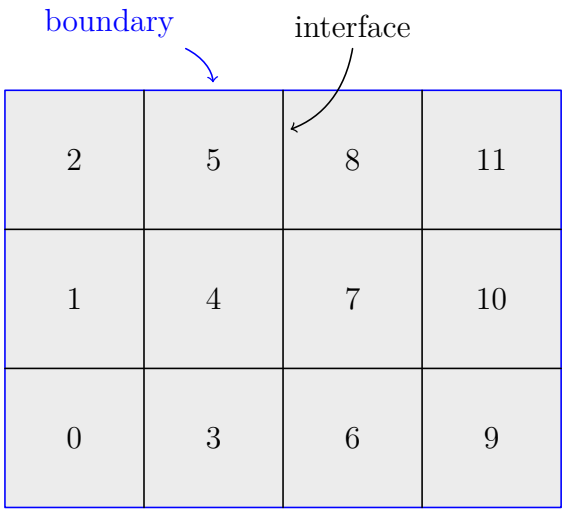
In the p-version of the finite element method (Szabo & Mehta, 1978; Babuska et al., 1981), the discretization is refined by increasing the polynomial degree, resulting in exponential convergence of the error with respect to the polynomial degree and the number of unknowns if the solution is smooth. This chapter introduces the basic ideas of the multi-level hp extension that follows in Chapter 4 in the context of single p-finite element meshes without refinements and with varying polynomial degrees per element and coordinate direction. This case is well known by the community and allows to introduce the multi-level hp extension to higher dimensions in a familiar setting.

3.1 Mesh and data structure

The meshes considered in this chapter consist of a set of *cells* that form a non-overlapping partition of the computational domain Ω. A cell is a d-dimensional (hyper-)cuboid, i.e., a line segment in one dimension, a quadrilateral in two dimensions, a cuboid or brick in three dimensions, and so on. A *finite element* is a cell that supports basis functions and is used to integrate the local finite matrices and vectors during the assembly of the global finite element equation system. Both terms are used interchangeably in this chapter in the context of p-finite elements. In Chapter 4, the parent cells in the hierarchically refined mesh are not finite elements, as they overlap with the leaf cells and are not considered separate entities during the assembly. Instead, their contributing shape functions are added to the non-overlapping leaf cells that are the multi-level hp-elements.

Each cell is assigned a unique *id* between 0 and $n_{\text{el}} - 1$. Figure 3.1a shows the cell numbering of an example mesh that is used throughout this chapter to demonstrate the construction of a p-finite element basis. While there is no specific restriction on the cell numbering, this particular choice seems natural as looping first over x (outer loop) and then over y (inner loop) leads to contiguous memory access of element-associated data. The same style of numbering the last dimension contiguously is also used in higher dimensions.

A cell has $2d$ local boundary *faces*, two per coordinate direction. Here, the term *face* is used generically to refer to the bounding topologies of codimension 1, which are two bounding nodes in one dimension, four bounding edges in two dimensions, six bounding faces in three dimensions, eight bounding cubes in four dimensions, and so on. These faces are the local cell boundaries that can either be part of the domain boundary $\partial\Omega$ or they can be *interfaces* to other cells. To prevent ambiguity, the term *boundary* refers to the domain (mesh) boundary and is not used in connection with local cell faces. The ids of both neighbor cells in d directions



(a) Cell (= element) numbering

Cell	n_l	n_r	n_b	n_t
0	-1	3	-1	1
1	-1	4	0	2
2	-1	5	1	-1
3	0	6	-1	4
4	1	7	3	5
5	2	8	4	-1
6	3	9	-1	7
7	4	10	6	8
8	5	11	7	-1
9	6	-1	-1	10
10	7	-1	9	11
11	8	-1	10	-1

(b) Left, right, bottom, and top neighbors

Figure 3.1: Example finite element mesh in two dimensions.

are stored in an adjacency array A of size $(n_{el}, d, 2)$, using a dedicated *no-cell* value for domain boundary faces; for example, -1 for signed integers or the maximum representable number for unsigned integers. Figure 3.1b shows the neighbor ids for all cells in 3.1a.

Unlike conventional p- and hp-finite element approaches, the mesh is not stored as a graph of vertices, edges, faces, and so on; instead, the algorithms are formulated to only use adjacency relations between cells. The main challenge in constructing C^0 continuous basis functions is to handle neighboring elements with varying polynomial degrees. To simplify the description of the algorithms, the coordinate axes of neighboring elements are assumed to be aligned, which is generally not true for unstructured meshes.

3.2 High-order shape functions[1]

This section discusses the tensor-products of integrated Legendre polynomials in the local (reference) coordinates $r_0, \ldots, r_{d-1}$, with $r_i \in [-1, 1]$, and their selective activation or deactivation. First, the integrated Legendre polynomials of degree q in one dimension are defined as

$$I_0(r) = \frac{1}{2}(1-r), \quad I_1(r) = \frac{1}{2}(1+r), \quad I_q(r) = \frac{P_q(r) - P_{q-2}(r)}{\sqrt{4q-2}} \quad \text{for } q > 1, \tag{3.1}$$

$$I_0'(r) = -\frac{1}{2}, \qquad I_1'(r) = \frac{1}{2}, \qquad I_q'(r) = \frac{P_q'(r) - P_{q-2}'(r)}{\sqrt{4q-2}} \quad \text{for } q > 1, \tag{3.2}$$

$$I_0''(r) = 0, \qquad I_1''(r) = 0, \qquad I_q''(r) = \frac{P_q''(r) - P_{q-2}''(r)}{\sqrt{4q-2}} \quad \text{for } q > 1, \tag{3.3}$$

where $P_q(r)$ are the Legendre polynomials of degree q:

$$P_0(r) = 1, \quad P_1(r) = r, \quad P_q(r) = \frac{2q-1}{q} r\, P_{q-1} - \frac{q-1}{q} P_{q-2} \qquad \text{for } q > 1,$$

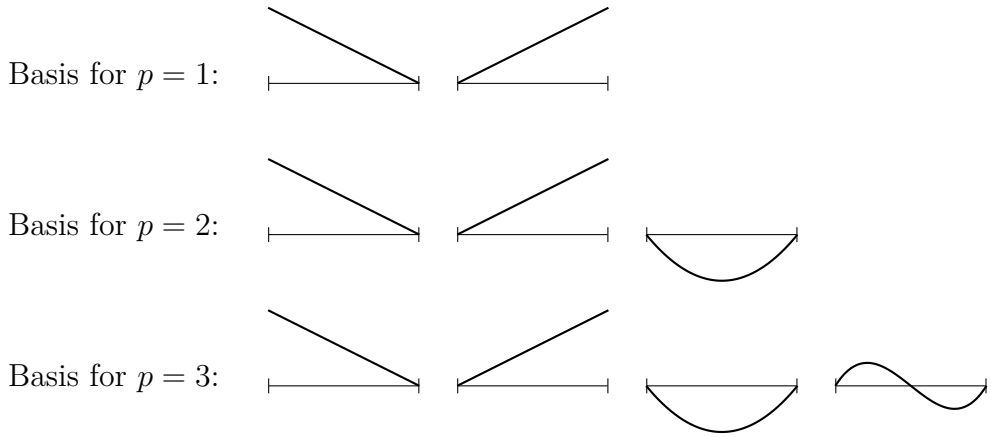


Figure 3.2: One-dimensional Integrated Legendre shape functions. From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

$$P_0'(r) = 0, \quad P_1'(r) = 1, \quad P_q'(r) = \frac{2q-1}{q}\Big(P_{q-1} + rP_{q-1}'\Big) - \frac{q-1}{q}P_{q-2}' \qquad \text{for } q > 1,$$

$$P_0''(r) = 0, \quad P_1''(r) = 0, \quad P_q''(r) = \frac{2q-1}{q}\Big(2P_{q-1}' + rP_{q-1}''\Big) - \frac{q-1}{q}P_{q-2}'' \quad \text{for } q > 1.$$

Due to several characteristics, integrated Legendre polynomials in this form are the standard choice in p-finite elements. First, I_0 is the only non-zero function on the left node and I_1 is the only non-zero function on the right node. All higher-order polynomials with $q > 1$ are zero on both nodes. A complete basis for a polynomial degree p is obtained by selecting the functions $\{I_0, \ldots, I_p\}$. These bases are hierarchical; *i.e.*, incrementing the polynomial degree adds only one function to the previous basis. Figure 3.2 sketches integrated Legendre bases for different polynomial degrees.

The construction of d-dimensional bases on a reference cube starts with the definition of integrated Legendre polynomials $\{I_0, \ldots, I_{p_a}\}$ for each coordinate axis $a \in \{0, \ldots, d-1\}$. The polynomial degrees in different coordinate axes may vary; thus, p is now a tuple $(p_0, \ldots, p_{d-1})$ with possibly different polynomial degrees per axis. This feature, for example, allows choosing different polynomial degrees in space and time for finite element discretizations of transient problems, or it can reduce the number of unknowns in combination with a capable error estimator. An element N_α in the tensor-product with the multi-index α

$$\alpha = (\alpha_0, \ldots, \alpha_{d-1}) \in \{0, \ldots, p_0\} \times \ldots \times \{0, \ldots, p_{d-1}\}, \tag{3.4}$$

is computed from the tensor-product of the univariate integrated Legendre polynomials in each coordinate direction:

$$N_\alpha(r) = \prod_{i=0}^{d-1} I_{\alpha_i}(r_i), \tag{3.5}$$

where r are the local coordinates r_0 until r_{d-1} for each direction. In two dimensions, for example, (3.5) becomes

$$N_{(\alpha_0, \alpha_1)}(r_0, r_1) = I_{\alpha_0}(r_0)\, I_{\alpha_1}(r_1) \qquad \text{for } \alpha_0 \in \{0, \ldots, p_0\} \text{ and } \alpha_1 \in \{0, \ldots, p_1\}, \tag{3.6}$$

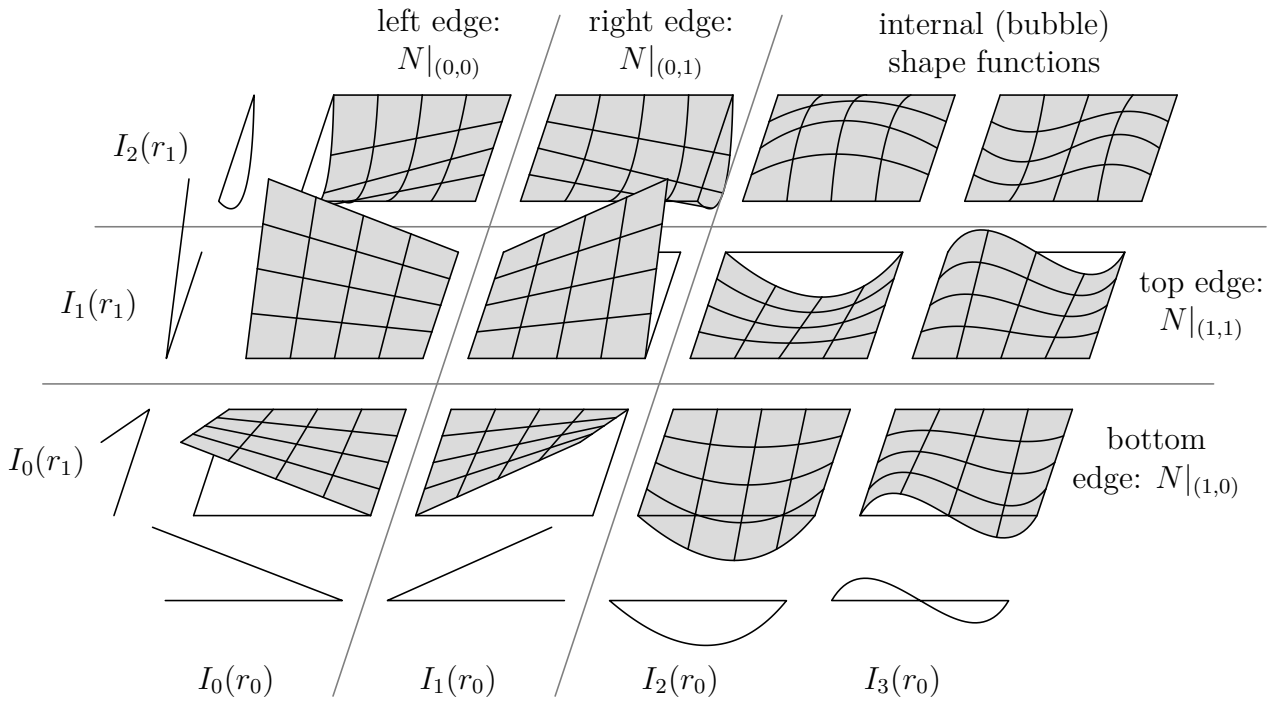


Figure 3.3: Tensor-product of integrated Legendre shape functions for $p = (3, 2)$. From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

where r_0 and r_1 are the two local coordinates. The functions in (3.5) and (3.6) are defined locally on each element, and they span complete polynomial spaces in the reference coordinate system. From now on, they are called shape functions to distinguish them from the finite element basis functions constructed later by mapping the shape functions of several adjacent elements from the reference to the global coordinates and "gluing" them together conformingly. Connecting or "gluing" shape functions together means assigning them the same global basis function index, as done by the algorithms of Sections 3.4 and 4.3.

Figure 3.3 displays the two-dimensional shape functions in (3.6) for $p = (3, 2)$. Independent of the polynomial degree, the first and second columns contain functions with support on the left and right element edges, and the first and second rows contain functions with support on the bottom and top edges. Similarly, in three dimensions, there are six two-dimensional subsets in the three-dimensional tensor-product, containing functions with support on each of the six faces.

This structure generalizes to a d-dimensional setting as follows. The tensor-product of the univariate integrated Legendre polynomials yields a d-dimensional array N that is indexed by the multi-index α, see (3.4) and (3.5). The functions with support on the first and second face in the coordinate direction a of the d-dimensional reference cube contain the factors $I_0(r_a)$ and $I_1(r_a)$, respectively. The subset of all such functions can be obtained by introducing an *array-slicing* operation that extracts a $d-1$-dimensional *array slice* (a subarray) from N. A specific array slice is identified by an integer pair (a, b), where $a \in \{0, \dots, d-1\}$ defines the fixed dimension (the axis the array slice is "orthogonal" to) and $b \in \{0, p_a - 1\}$ the index along the axis a. With this general definition, elements in the array slice $N|_{(a,b)}$ contain the elements of N_α for which $\alpha_a = b$. In the context of this thesis, the only interesting array slices

are those with $b = 0$ and $b = 1$, as they extract the shape functions with support on the two faces in the direction a.

For example, in two dimensions, the array slice $N|_{(0,1)}$ is orthogonal to axis zero at index one, and it contains the shape functions active on the right edge. Similarly, the array slice $N|_{(1,0)}$ is orthogonal to axis one at index zero, and it contains the functions active on the bottom edge. The remaining two array slices, $N|_{(0,0)}$ and $N|_{(1,1)}$, extract the shape functions on the left and top edges. In three dimensions, the six array slices $N|_{(0,0)}$, $N|_{(0,1)}$, $N|_{(1,0)}$, $N|_{(1,1)}$, $N|_{(2,0)}$, and $N|_{(2,1)}$ contain the functions with support on the left, right, front, back, bottom, and top faces, respectively. In four dimensions, there are eight three-dimensional array slices for each of the eight cubes, and so on. Together with the hierarchy in p, this framework forms the basis for algorithmically connecting the shape functions into p- and hp-basis functions. For example, in Section 4.2, homogeneous Dirichlet conditions are imposed on the boundaries of overlay cells by simply deactivating the corresponding array slices in the tensor-product. The presented algorithms extend well to higher dimensions because they do not introduce a further classification of the shape functions into nodal modes, edge modes, face modes, and so on, unlike traditional p- and hp-finite element approaches.

Remark 3.2.1 *The integrated Legendre polynomials* (3.1) *can be expanded and evaluated using Horner's scheme. However, the recursive definition is computationally more efficient when computing all shape functions $I_0 - I_p$ at once.*

3.3 Tensor-product masks[1]

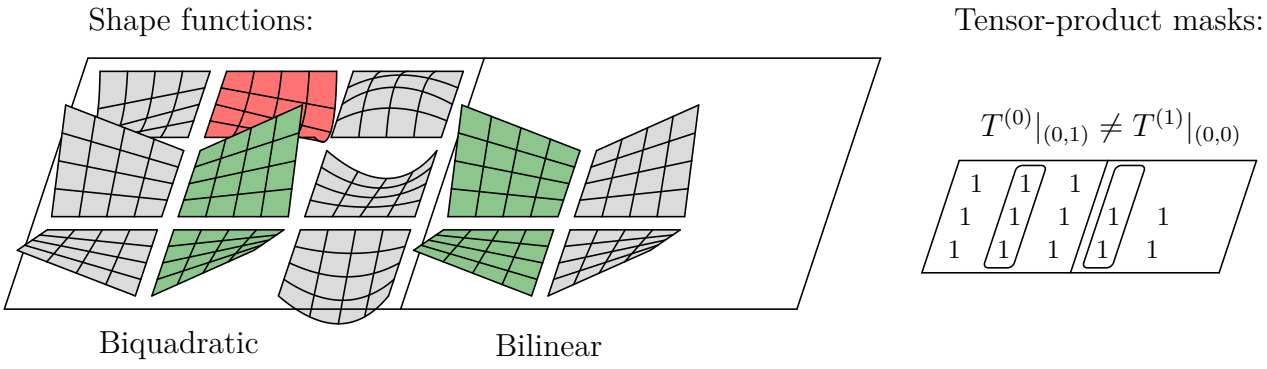


Figure 3.4: Second-order edge mode (shaded in red) in the quadratic left element is not present in the linear right element, rendering the shape function sets of both elements incompatible.

In p-finite elements, the polynomial degree can be chosen independently on each element and in each local direction. This introduces the challenge of having non-matching sets of shape functions between elements with different polynomial degrees in the tangential directions on the interface. For example, a vertical edge in two dimensions may have two adjacent elements with different polynomial degrees in the vertical direction. When using hierarchical polynomials, the basis with the lower degree is contained by the one on the other side of the shared edge. As a result, the matching shape functions on the shared edge (up to the highest common polynomial degree) can directly be connected to form global C^0 basis functions. The

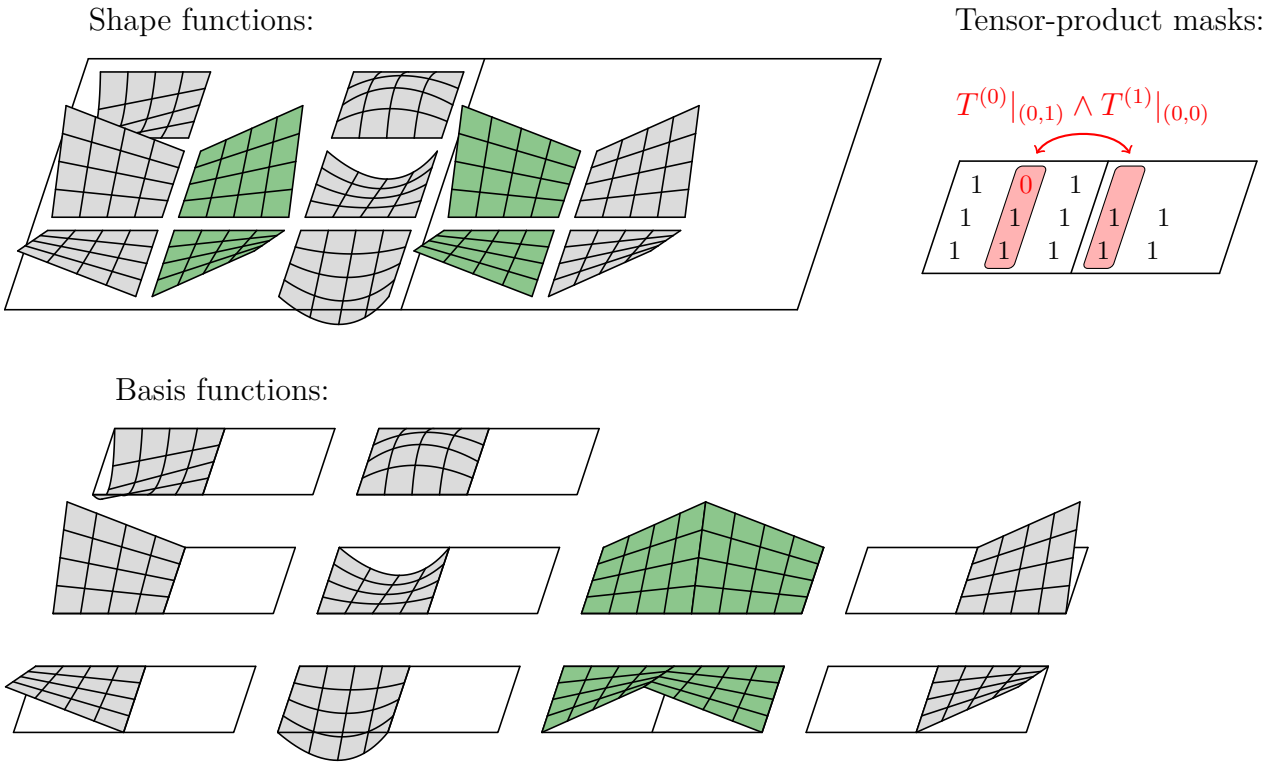


Figure 3.5: Remove left shape function without counterpart on the right element and only connect the remaining two pairs of bilinear functions.

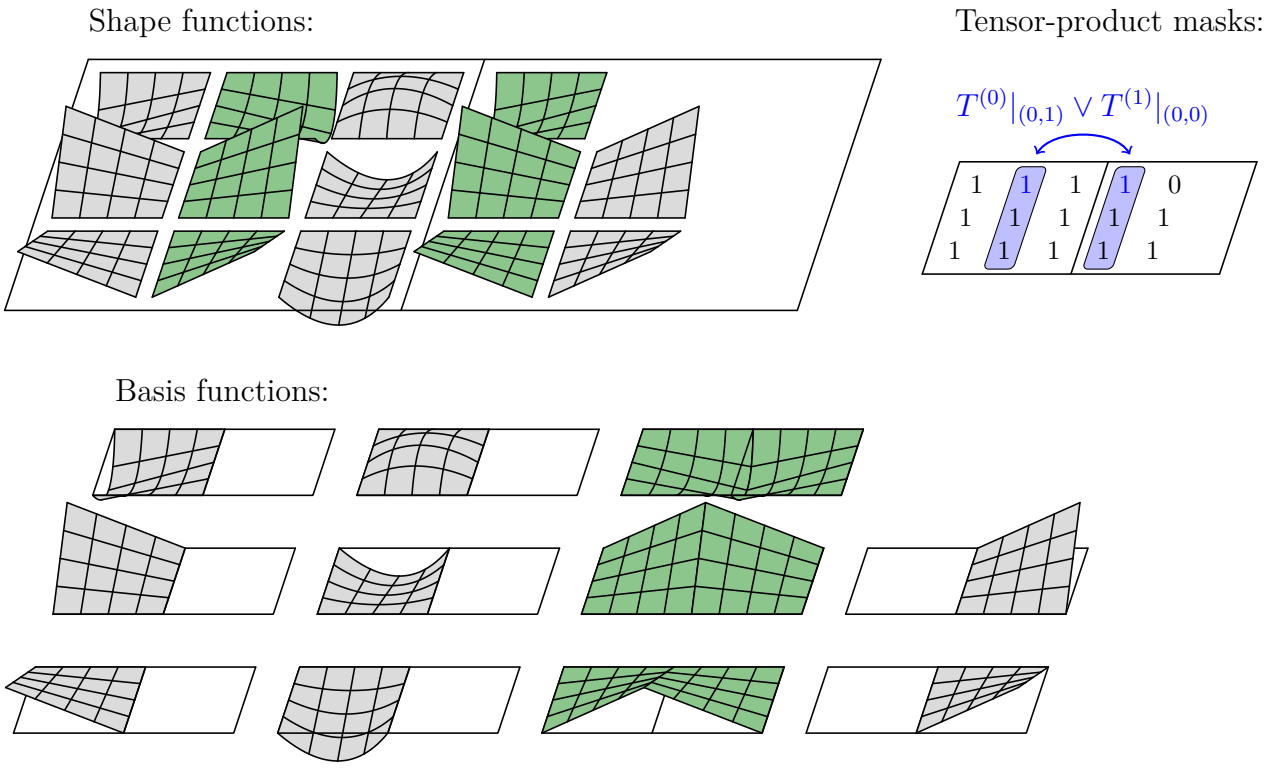


Figure 3.6: Add corresponding second-order shape function on the right element and connect three shape function pairs with support on the shared edge into three (global) basis functions.

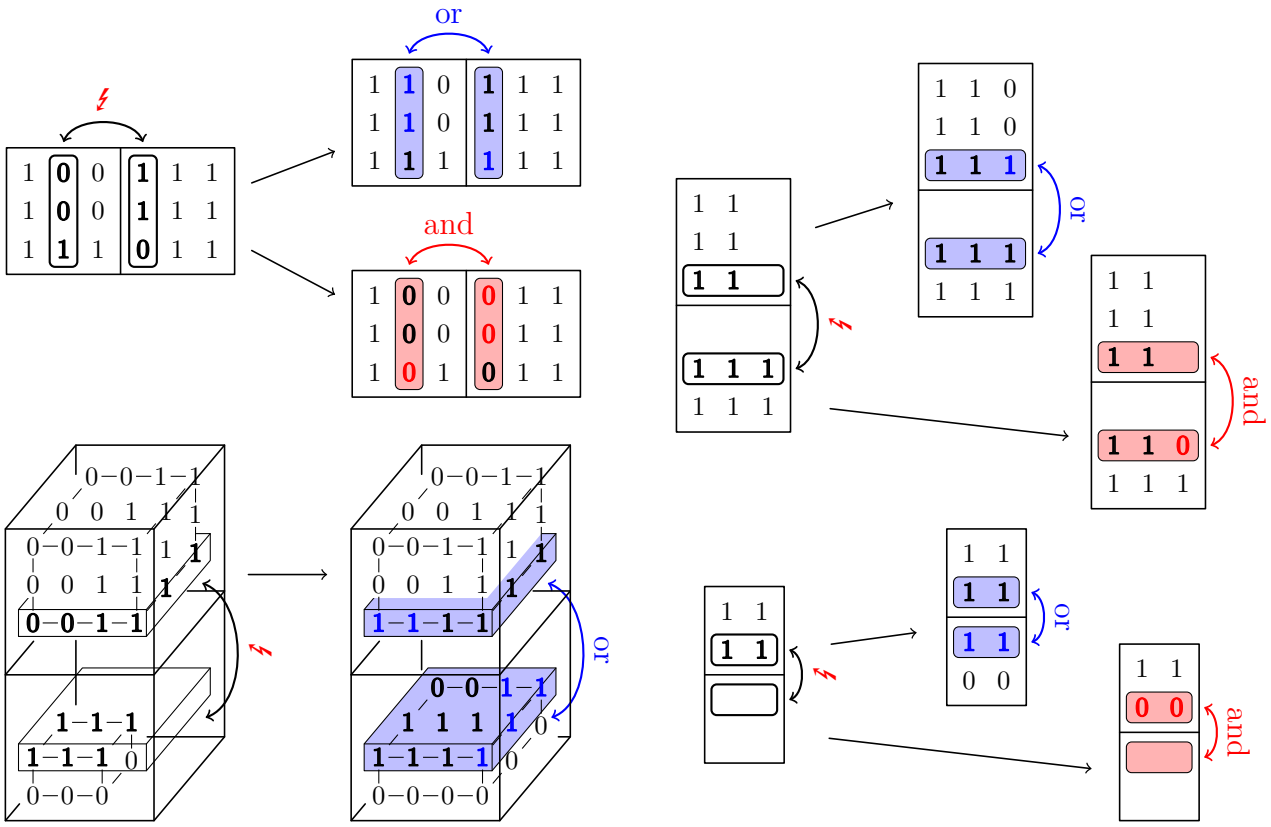


Figure 3.7: Restore interface compatibility between the tensor-product masks of two cells by comparing and overwriting the interface slices using logical operations. Array slices to be compared are printed in boldface; blue entries were activated due to logical *or*; red entries were deactivated due to logical *and*. From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

left side of Figure 3.4 shows the tensor-product shape functions of two elements: the left with $p = 2$ and the right element with $p = 1$, both with uniform polynomial order in both directions. The set of shape functions on the left element contains a quadratic mode, which is not present in the right element, as indicated by the red color. The two linear shape functions have a corresponding counterpart in the other element and are therefore colored in green. There are two options for recovering two matching sets of shape functions: either the quadratic function is removed on the left element, or the corresponding function is added in the right element. In general, one can deactivate the additional functions on the element with a higher degree (minimum degree strategy). Alternatively, new shape functions can be added to the element with a lower degree (maximum degree strategy). Figures 3.5 - 3.6 apply these two options to the example shown in Figure 3.4 and display the resulting basis functions after connecting the matching shape functions.

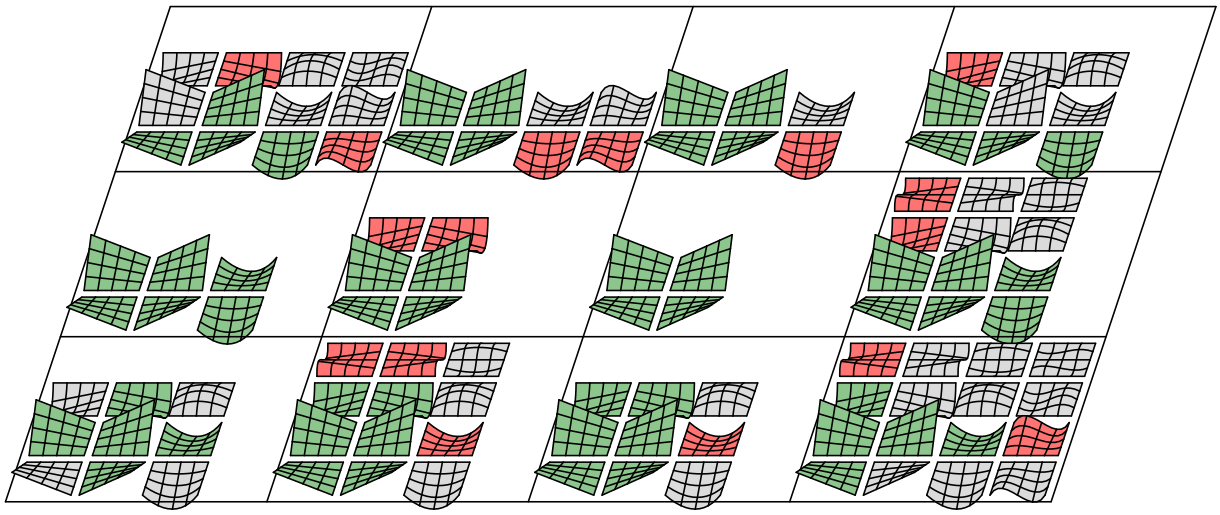
This section details a data structure and an algorithm for obtaining sets of shape functions for each finite element that are compatible across the element interfaces. Compatible means that all shape functions that are non-zero on the element faces have a matching counterpart in the neighbor, which allows them to be connected to C^0 continuous basis functions in Section 3.4 (assigned to the same global basis function index). The data structure for storing the activation

$(3,2)$	$(3,1)$	$(2,1)$	$(2,2)$
$(2,1)$	$(1,2)$	$(1,1)$	$(2,3)$
$(2,2)$	$(2,3)$	$(2,2)$	$(3,3)$

(a) Polynomial degrees

1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1	1 1 1 1 1 1 1 1 1
1 1 1 1 1 1	1 1 1 1 1 1	1 1 1 1	1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

(b) Initial tensor-product masks

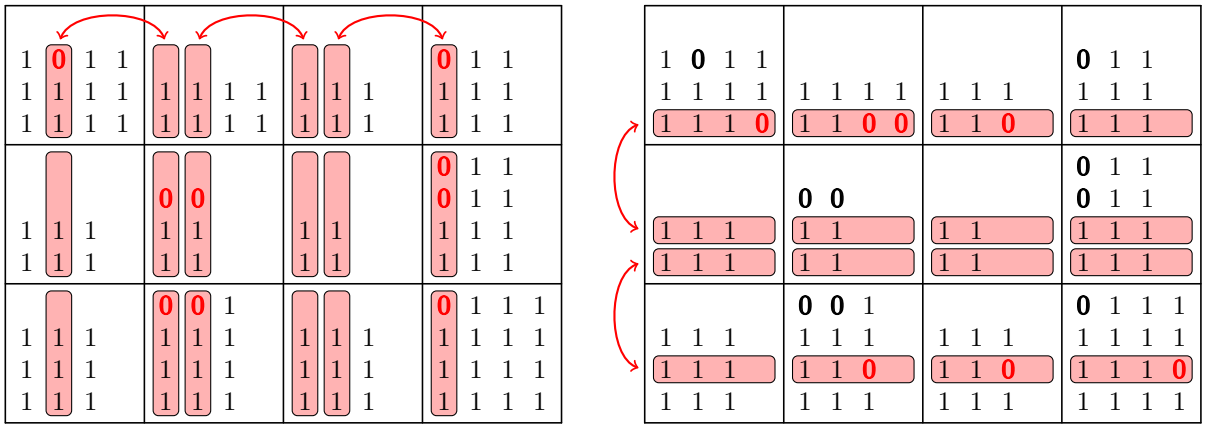


(c) Internal (gray), incompatible (red) and matching (green) shape functions in (b).

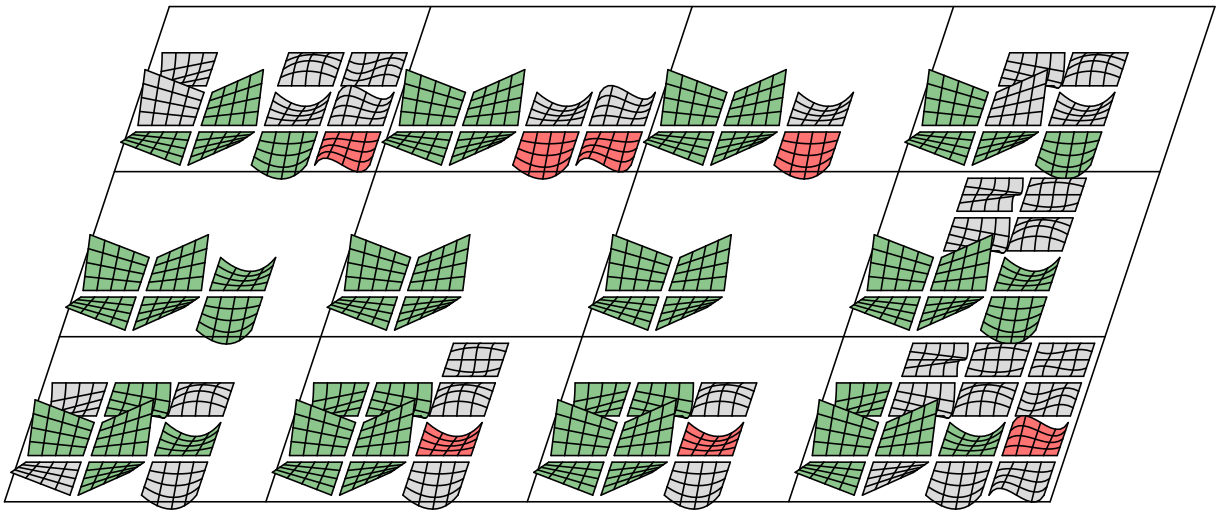
Figure 3.8: Initial activation of all p-finite element shape functions independent of the neighbors. From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

state of the tensor-product shape functions consists of a *tensor-product mask* for each finite element. A tensor-product mask $T^{(e)}$ of an element with index e is a d-dimensional array of Boolean entries $T^{(e)}_\alpha$ that determine whether the corresponding shape function $N_\alpha(x)$ from the tensor-product (3.5) is active in element e. Each tensor-product mask must be large enough to store all active values; entries beyond are implicitly assumed zero. For a classical p-finite element mesh in two dimensions with polynomial degree p in both directions, the tensor-product masks become $(p+1) \times (p+1)$ sized matrices with only active entries.

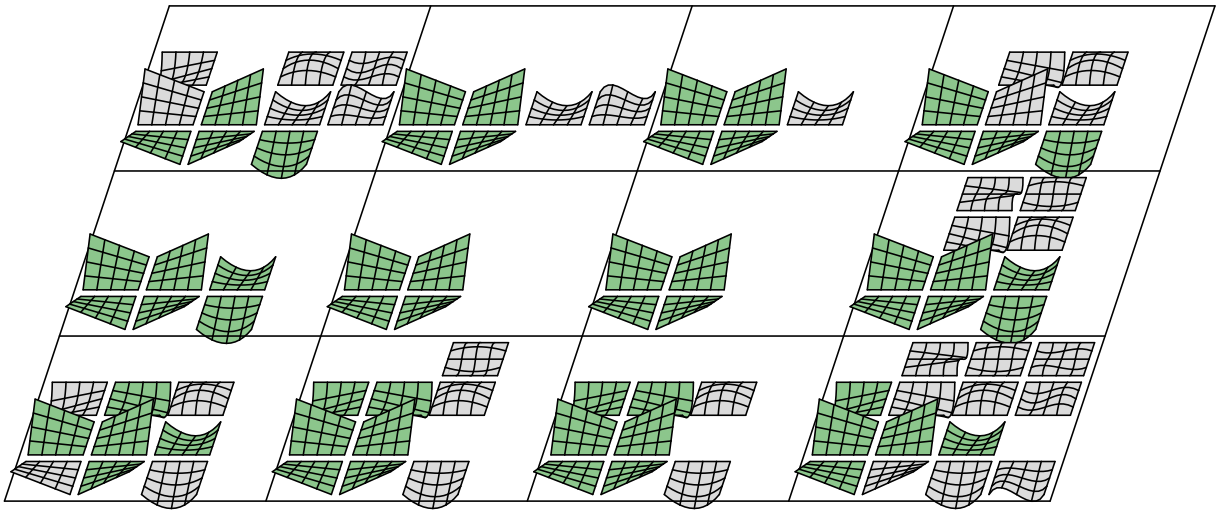
The fully activated tensor-product masks are incompatible between elements with different polynomial degrees on a shared interface. The right side of Figure 3.4 shows the two incompatible tensor-product masks for the elements with $p = 2$ and $p = 1$. They must be modified to recover a state where pairs of functions from each element with support on the shared interface can be combined to globally C^0 continuous basis functions. This can be done by either deactivating entries if they are not present in the neighbor (right side of Figure 3.5) or activating entries in the neighbor if they are present in this element (right side of Figure 3.6).



(a) Make tensor-product masks compatible using logical *and*: first over x-, then over y-interfaces

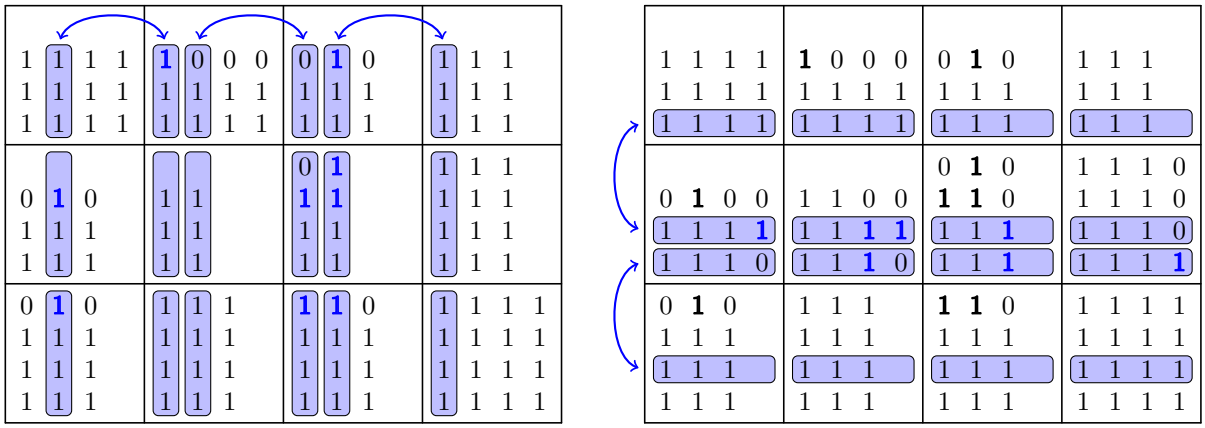


(b) Intermediate state after operating on x-interfaces (of Figure 3.8c) using logical *and*

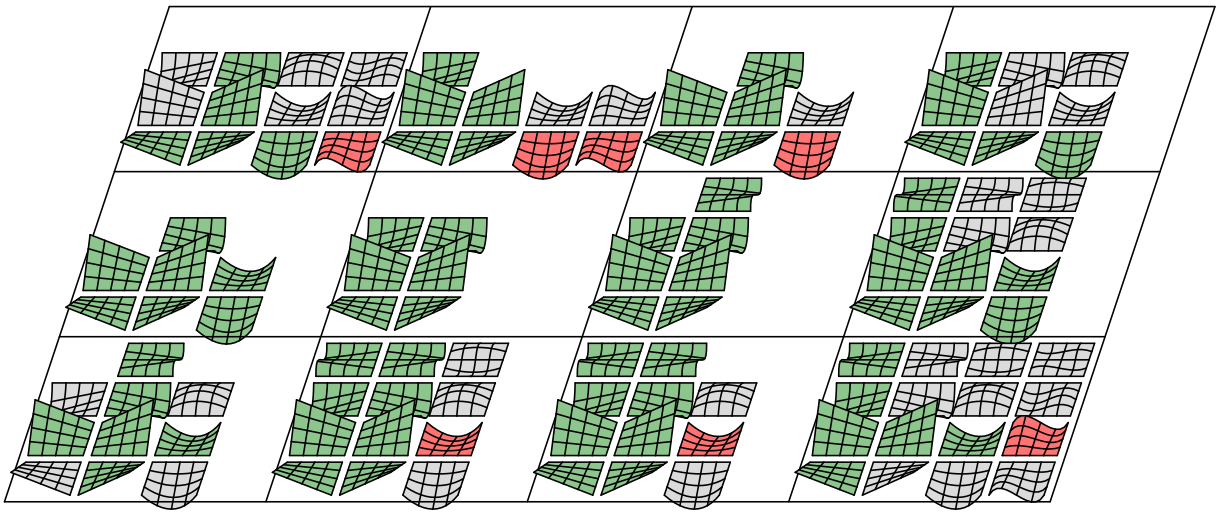


(c) Final shape functions after operating on y-interfaces (of b) using logical *and*

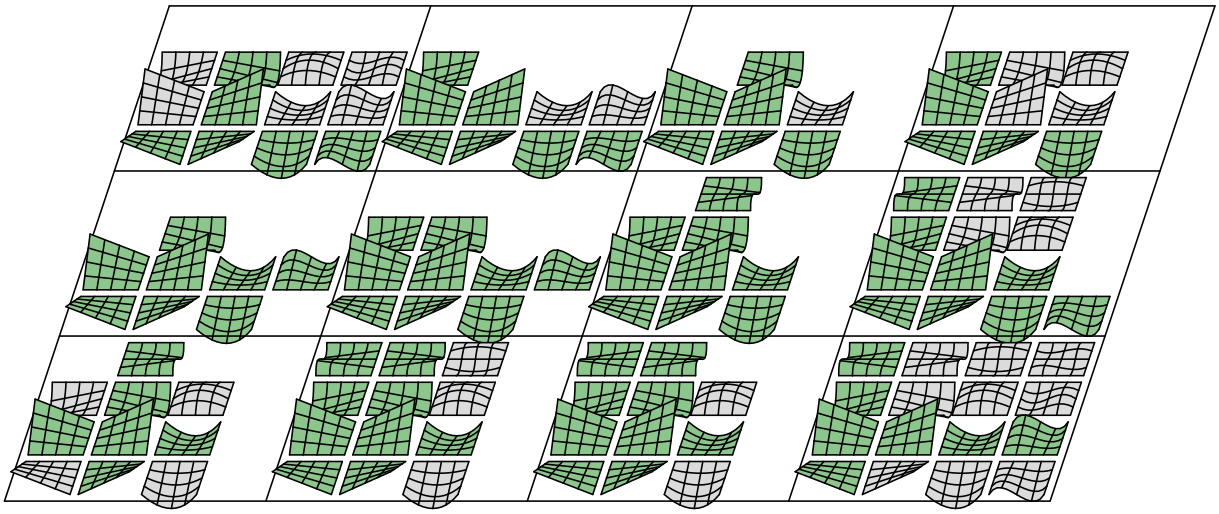
Figure 3.9: Selectively deactivate shape functions using logical *and* operations on the element-associated tensor-product masks to obtain compatible sets of shape functions.



(a) Make tensor-product masks compatible using logical *or*: first over x-, then over y-interfaces



(b) Intermediate state after operating on x-interfaces (of Figure 3.8c) using logical *or*



(c) Final shape functions after operating on y-interfaces (of b) using logical *or*

Figure 3.10: Selectively deactivate shape functions using logical *or* operations on the element-associated tensor-product masks to obtain compatible sets of shape functions.

Algorithm 1 Construct compatible tensor-product masks for single p-finite element mesh.

```
1  // Compare the array slices (n,1) and (n,0) of the nd-arrays A_0 and A_1 using oper-
2  // ation op and overwrite them with the result. Data types are either bool or int.
3  void operateOnInterface(NdArray<Type>& A_0, NdArray<Type>& A_1, int n, Op op)
4  {
5      // Extract sizes of arrays in each direction without entry at index n
6      Vector s_0 = sizes(A_0);
7      Vector s_1 = sizes(A_1);
8
9      Vector s_0^n = removeEntry(s_0, n);
10     Vector s_1^n = removeEntry(s_1, n);
11
12     // Compute element-wise maximum s^n of s_0^n and s_1^n and iterate over all
13     // entries in the Cartesian product [0, s^n(0)] × ... × [0, s^n(d - 2)]
14     for(Vector i^n in productIndices(maxarray(s_0^n, s_1^n)))
15     {
16         // Insert into index tuple i^n in axis direction
17         Vector i_0 = insertEntry(i^n, n, 1); // second face of first element
18         Vector i_1 = insertEntry(i^n, n, 0); // first face of second element
19
20         bool e_0 = indexExists(i_0, s_0);
21         bool e_1 = indexExists(i_1, s_1);
22
23         Type v_0 = A_0(i_0) if e_0 else noValue(Type);
24         Type v_1 = A_1(i_1) if e_1 else noValue(Type);
25
26         Type r = op(v_0, v_1);
27
28         if(v_0 ≠ r and not e_0) resize(A_0, i_0);
29         if(v_1 ≠ r and not e_1) resize(A_1, i_1);
30
31         if(v_0 ≠ r) A_0(i_0) = r;
32         if(v_1 ≠ r) A_1(i_1) = r;
33     }
34 }
35
36 // Compare array slices of all cells with second neighbor if it exists
37 void operateOnInterfaces(NdArrayList<Type>& A, Neighbors N, Op op)
38 {
39     for(int a from 0 to d - 1)          // Loop over coordinate axes
40         for(int i from 0 to size(N))    // Loop over cell indices
41             if(int N_a = N(i,a,1); N_a ≠ -1)
42                 operateOnInterface(A(i), A(N_a), a, op);
43 }
44
45 MaskList createPfemMasks(Neighbors N, Degrees p, String strategy)
46 {
47     MaskList M;
48
49     // Activate full tensor-product (could use trunk space masks instead)
50     for(Vector p_e in p)
51         M.append(Mask(p_e + 1, true));
52
53     Op op = logicalAnd if strategy == "min_degree" else logicalOr;
54
55     // Recover compatibility
56     for(int it from 0 to d - 2)
57         operateOnInterfaces(M, N, op);
58
59     return M;
60 }
```

The first strategy yields the minimum polynomial degree on the interface, and the second yields the maximum. The mask of the right element in Figure 3.6 was enlarged to match the higher polynomial degree of the neighbor on the interface.

Both strategies can be expressed in logical operations on the $d-1$ dimensional interface slices in the tensor-product masks of both elements. Similar to the previous section, these operations use array slices to manipulate the Boolean mask entries corresponding to the shape functions with support on the faces of the reference cube. In this context, an *interface slice* refers to an array slice of a face with an existing neighbor (the face is not part of the domain boundary). In two dimensions, these interface slices are columns or rows in the Boolean matrix that are compared element-wise using a logical *and* to obtain the minimum-degree strategy (Figure 3.5) or a logical *or* to obtain the maximum-degree strategy (Figure 3.6). The result of the logical operation is then written back to both original interface slices. Moreover, these logical operations rely on the property that elements outside the size of a mask are implicitly inactive. Figure 3.7 shows more examples of logical operations on interface slices for two- and three-dimensional elements. The logical *or* operation may again require enlarging some of the tensor-product masks. Empty masks like those on the bottom right may occur in the hierarchical version in Section 4.2.

In this setting, the tensor-product masks for p-finite element meshes are constructed in two steps. First, all masks are initialized by activating the full tensor-product up to the specified polynomial degrees. Figure 3.8 defines tuples of polynomial degrees for the mesh of Figure 3.1, shows the resulting (incompatible) initial tensor-product masks, and visualizes the corresponding shape functions. In the second step, compatibility is recovered on all element interfaces using either logical *and* operations for obtaining minimum interface degrees or logical *or* operations for maximum degrees. Figures 3.9 - 3.10 demonstrate these two variants to make the initial tensor-product masks of Figure 3.8 compatible. The algorithm first loops over all interfaces in x- and then in y-direction. Some pairs of tensor-product masks have different sizes in the interface direction, which uses the definition that entries are inactive if they are not explicitly stored. In d dimensions, the second step is carried out $d-1$ times to reach all elements that the shape functions connect to. For example, in three dimensions, the basis functions associated with an edge may be supported by four elements; hence, information needs to travel diagonally by communicating over two interfaces. The second step of Section 4.2 may clarify this information transfer as already two iterations are necessary in two dimensions. Algorithm 1 shows the construction of tensor-product masks for a d-dimensional mesh.

One major difference compared to other p- and hp- finite element approaches is that they often use data structures with unique entities for all topological subcomponents, such as nodes, edges, and faces, with their associated polynomial degrees. For example, a unique edge may be introduced for each pair of adjacent elements in a two-dimensional p-finite element mesh, where the associated polynomial degree in the tangential direction is chosen from one of the two elements. However, the resulting data structure becomes drastically more complex with higher dimensions (containing 3^d entities per element in total). There, the runtime and memory requirements for creating and maintaining meshes may become dominant. Moreover, evaluating the basis functions requires considering all subcomponents and their respective polynomial degrees per local direction, which complicates the implementation and can also negatively influence the performance of the linear system assembly.

Algorithm 2 Construct location matrices for a single p-finite element mesh.

```
1  [LocationMatrixList, int] initializeGlobalIndices(MaskList M)
2  {
3      LocationMatrixList G; int n_ids = 0;
4
5      for(int i from 0 to size(M) - 1)
6      {
7          Mask M_i = M(i);
8          LocationMatrix G_i(sizes(M_i));
9
10         // Loop over all tensor-product entries (d nested loops)
11         for(Vector j in productIndices(sizes(M_i)))
12             G_i(j) = n_ids++ if M_i(j) else -1;
13
14         G.append(G_i);
15     }
16
17     return [G, n];
18 }
19
20 void removeUnassignedIndices(LocationMatrixList& G, int n_ids)
21 {
22     Vector exists(n_ids, 0);
23     Vector map(n_ids, -1);
24
25     // Activate entries in exists vector
26     for(LocationMatrix G_i in G)
27         for(Vector j in productIndices(sizes(G_i)))
28             if(G_i(j) ≠ -1)
29                 exists(G_i(j)) = 1;
30
31     int n_new = 0;
32
33     // Count active dofs upwards into map
34     for(int i from 0 to nids - 1)
35         map(i) = n_new++ if exists(i) else −1;
36
37     // Reassign indices in location matrices
38     for(LocationMatrix& G_i in G)
39         for(Vector j in productIndices(sizes(G_i)))
40             if(G_i(j) ≠ -1)
41                 G_i(j) = map(G_i(j))
42 }
43
44 // Create location matrix with global ids for each tensor-product mask
45 void createPfemLocationMatrices(MaskList M, Neighbors N)
46 {
47     LocationMatrixList G, int n_ids = initializeGlobalIndices(M);
48
49     for(int it from 0 to d - 1)
50         operateOnInterfaces(G, N, minimum);
51
52     removeUnassignedIndices(G, n_ids);
53
54     return G;
55 }
```

17 -1 22 24 16 19 21 23 15 18 20 -1	39 41 42 43 38 40 -1 -1	57 59 60 56 58 -1	-1 89 92 86 88 91 85 87 90
10 12 14 9 11 13	-1 -1 35 37 34 36	53 55 52 54	-1 80 84 -1 79 83 76 78 82 75 77 81
2 5 8 1 4 7 0 3 6	-1 -1 33 27 30 32 26 29 -1 25 28 31	46 49 51 45 48 -1 44 47 50	-1 67 71 74 63 66 70 73 62 65 69 -1 61 64 68 72

(a) Assign unique indices

17 -1 22 24 16 19 21 23 15 18 20 -1	19 41 42 43 18 40 -1 -1	41 59 60 40 58 -1	-1 89 92 59 88 91 58 87 90
10 12 14 9 11 13	-1 -1 12 37 11 36	37 55 36 54	-1 80 84 -1 79 83 55 78 82 54 77 81
2 5 8 1 4 7 0 3 6	-1 -1 33 5 30 32 4 29 -1 3 28 31	30 49 51 29 48 -1 28 47 50	-1 67 71 74 49 66 70 73 48 65 69 -1 47 64 68 72

(b) Connect in x

17 -1 22 24 16 19 21 23 10 12 14 -1	19 41 42 43 12 37 -1 -1	41 59 60 37 55 -1	-1 89 92 59 88 91 55 78 82
10 12 14 1 4 7	-1 -1 12 37 4 29	37 55 29 48	-1 80 84 -1 79 83 55 78 82 48 65 69
2 5 8 1 4 7 0 3 6	-1 -1 33 5 30 32 4 29 -1 3 28 31	30 49 51 29 48 -1 28 47 50	-1 67 71 74 49 66 70 73 48 65 69 -1 47 64 68 72

(c) Connect in y

13 -1 16 18 12 14 15 17 9 10 11 -1	14 26 27 28 10 25 -1 -1	26 35 36 25 34 -1	-1 55 57 35 54 56 34 48 51
9 10 11 1 4 7	-1 -1 10 25 4 20	25 34 20 30	-1 50 53 -1 49 52 34 48 51 30 38 42
2 5 8 1 4 7 0 3 6	-1 -1 24 5 21 23 4 20 -1 3 19 22	21 31 33 20 30 -1 19 29 32	-1 40 44 47 31 39 43 46 30 38 42 -1 29 37 41 45

(e) Eliminate unassigned indices using (d)

old index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
exists:	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0	1	1	0	1	0	1	1	1
new index:	0	1	2	3	4	5	6	7	8	-1	9	-1	10	-1	11	-1	12	13	-1	14	-1	15	16	17
old index:	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
exists:	1	0	0	0	1	1	1	1	1	1	0	0	0	1	0	0	0	1	1	1	0	0	0	1
new index:	18	-1	-1	-1	19	20	21	22	23	24	-1	-1	-1	25	-1	-1	-1	26	27	28	-1	-1	-1	29
old index:	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71
exists:	1	1	1	1	0	0	0	1	0	0	0	1	1	0	0	0	1	1	1	1	1	1	1	1
new index:	30	31	32	33	-1	-1	-1	34	-1	-1	-1	35	36	-1	-1	-1	37	38	39	40	41	42	43	44
old index:	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92			
exists:	1	1	1	0	0	0	1	1	1	0	1	1	1	0	0	0	1	1	0	1	1			
new index:	45	46	47	-1	-1	-1	48	49	50	-1	51	52	53	-1	-1	-1	54	55	-1	56	57			

(d) Index map

Figure 3.11: Glue shape functions from Figure 3.9a together into compatible basis functions. From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

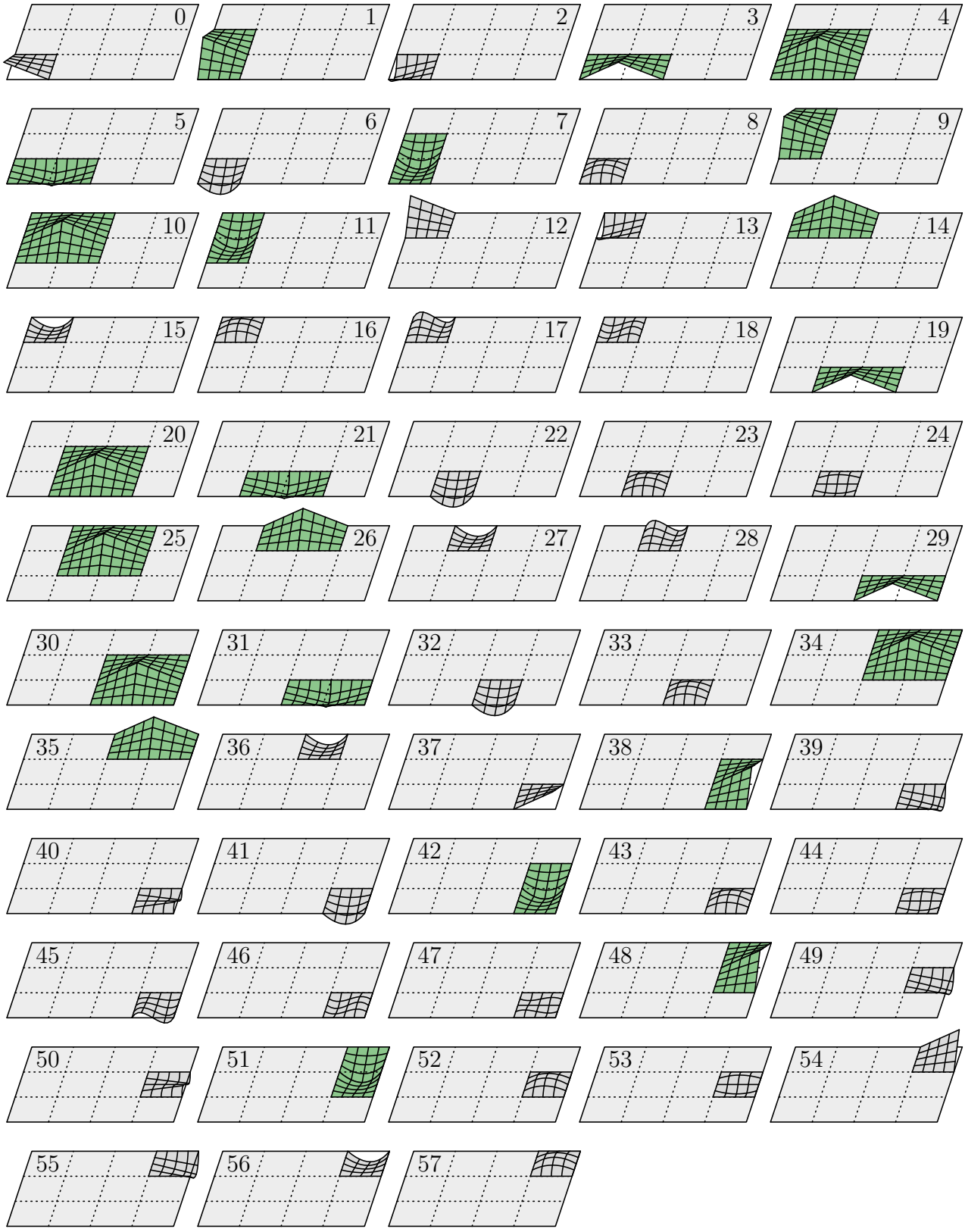


Figure 3.12: Connected p-finite element basis functions according to Figure 3.11e.

3.4 Location matrices[1]

The tensor-product masks constructed in the previous section determine the active shape functions for all elements, such that they have matching counterparts across all interfaces. The remaining task is to connect sets of shape functions from different elements into basis functions by assigning them to the same global basis function index. This data is stored by introducing a *location matrix* for each tensor-product mask with equal size containing the global basis function index for each active entry or otherwise some value, e.g., -1 or the maximum representable number for inactive entries. In higher dimensions, the location matrices are d-dimensional arrays containing the basis function indices.

The algorithm to populate the location matrices starts by assigning a unique global index to all shape functions, counting upwards from zero. Figure 3.11a shows the initialization of the location matrices for the tensor-product masks of the minimum-degree strategy (right side of Figure 3.9a). Then, matching shape functions are connected by looping over all element interfaces, comparing the corresponding interface slices in the location matrices using the *minimum* operation (selecting the smaller of both indices), and overwriting them with the result. The only difference to the interface operations in the second step of Section 3.3 is the data type (integers instead of Boolean values) and the operation (*minimum* instead of logical *and* or logical *or*). This step automatically determines the smallest index for each set of shape functions that shall be connected and propagates this information back to their entries in the location matrices. Figures 3.11b and 3.11c demonstrate this process by connecting the indices in the x- and then in the y-direction. Mesh topologies that are not Cartesian (e.g., an L-shaped domain with axis-aligned elements) may require again multiple iterations of this step. This step leaves some empty indices (i.e., without contributions from any elements), as their original shape functions have been assigned to a different index. These "gaps" in the global numbering are eliminated by constructing a new set of compressed indices in Figure 3.11e that is obtained by applying the index map below. Figure 3.12 shows the resulting basis functions for this example. Algorithm 2 summarizes the creation of the location matrices and thus finalizes the construction of the p-finite element basis.

Remark 3.4.1 *Linearizing a location matrix and filtering inactive entries (-1 values) yields a global index vector called location maps or an element freedom table.*

3.5 Trunk space[1]

Section 3.3 mentions that using all functions from the tensor-product is not always necessary. Within a monomial basis,

$$\left\{ r_0^{\alpha_0} \cdot \ ... \ \cdot r_{d-1}^{\alpha_{d-1}} \;\middle|\; \text{for } \alpha_i = 0, ..., p_i \right\}, \tag{3.7}$$

one can pick only functions where $\sum \alpha_i \leq \max p_i$ without reducing the convergence order. Figure 3.13 shows the monomials for a tensor-product space with $p = (3, 4)$ compared to the reduced basis containing only polynomials with a combined order smaller or equal to $\max p_i = 4$.

As monomials are unsuitable for constructing compatible finite element bases, this concept is transferred to integrated Legendre functions. In Section 3.3, the algorithm for compatible shape functions started with a full tensor-product mask for leaf cells. Instead, one can

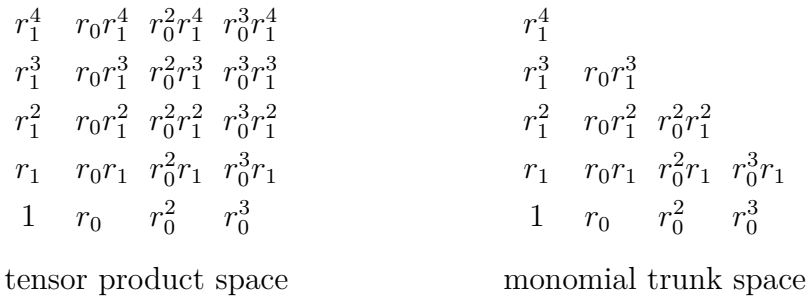


Figure 3.13: Monomials for $p = (3, 4)$. From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

construct initial trunk space masks by only activating entries of $M_{\boldsymbol{\alpha}}$ if $\sum \alpha_i \leq \max p_i$. The integrated Legendre functions are hierarchical except in the first two functions (I_0 and I_1 are linear). As a result, a function with component $I_0(r_i)$ must be inactive, when the corresponding function with component $I_1(r_i)$ is inactive. Therefore, the first array slice for each coordinate axis is copied to the second array slice. Figures 3.14 and 3.15 demonstrate this procedure on two- and three-dimensional examples.

The trunk space significantly reduces the number of unknowns compared to a tensor-product space of the same polynomial degree without compromising the convergence order. Removing basis functions that are not essential for polynomial completeness (and, thus, the convergence order) decreases the computational cost significantly. However, this reduces the accuracy of the solution at the same time. When increasing the polynomial degree to compensate, one generally still obtains fewer degrees of freedom for a given accuracy at the cost of having higher connectivity per unknown. With higher dimensions, the difference between them increases as

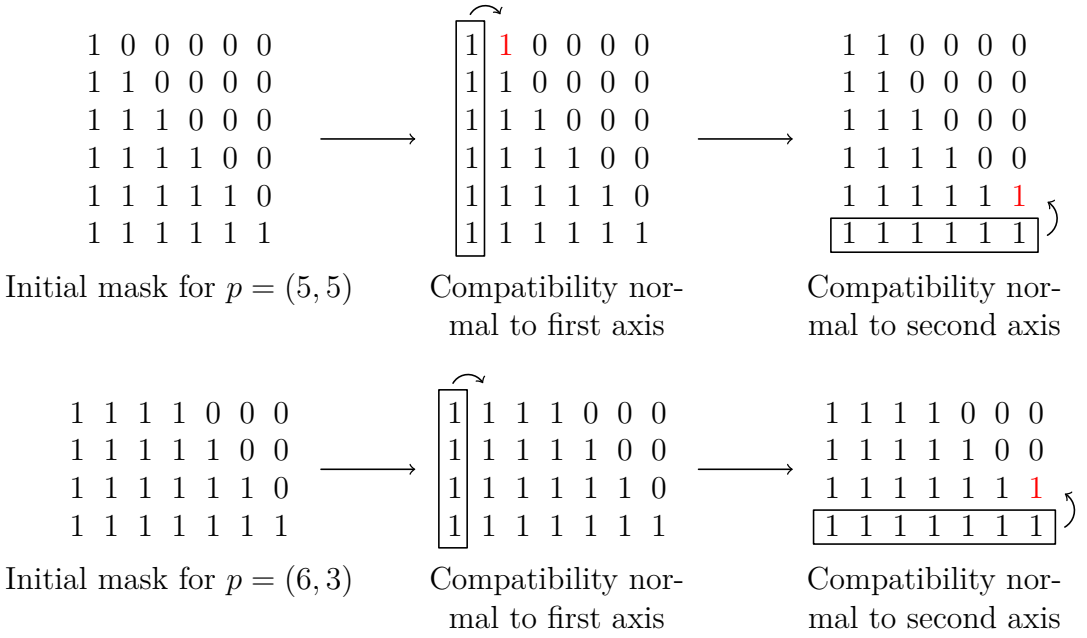


Figure 3.14: Construction of two-dimensional trunk space initial tensor-product masks. From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

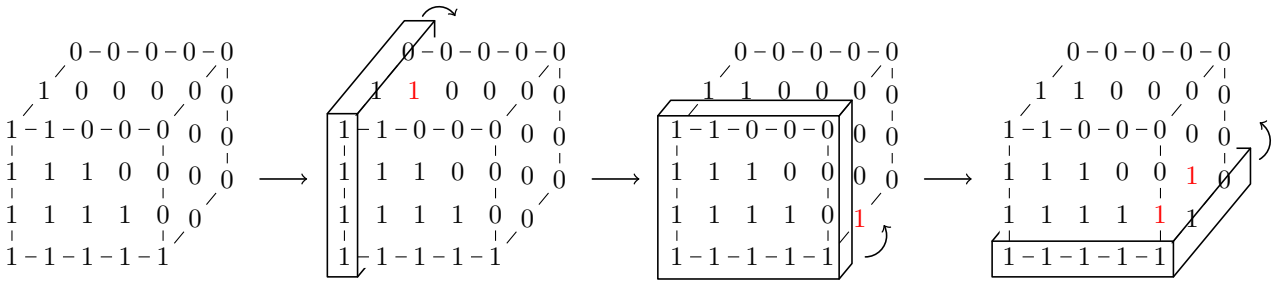
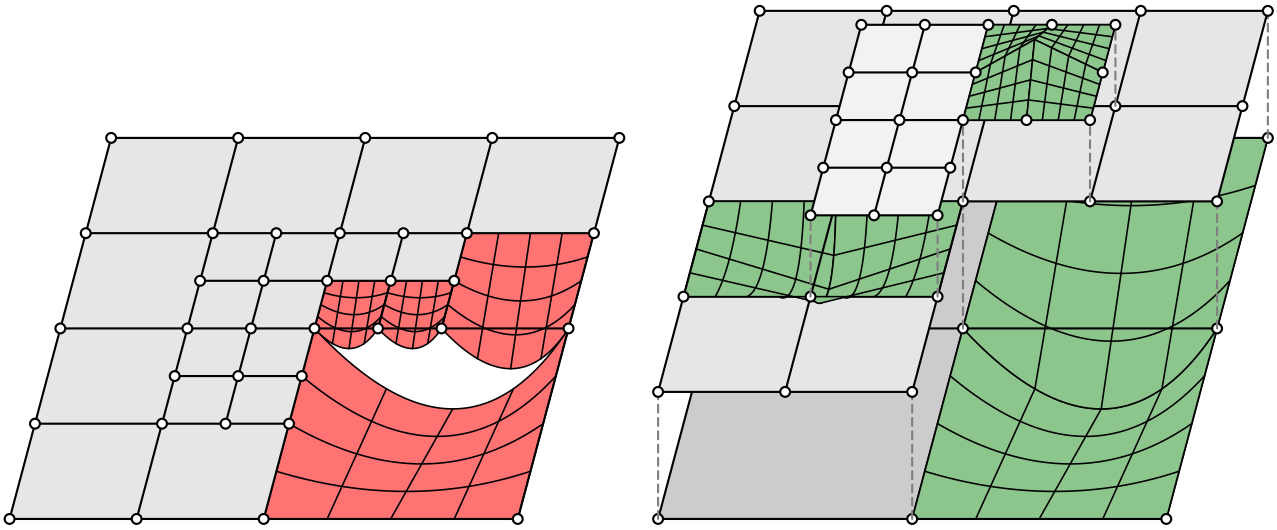


Figure 3.15: Construction of three-dimensional trunk space initial tensor-product masks. From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

the limit for $p \to \infty$ yields $d!$ times fewer unknowns per element (assuming uniform p in all directions). This limit derives from considering an n-simplex in d-dimensional space with one vertex at the origin and one vertex on each coordinate axis with distance 1. The volume of this simplex is $1/d!$, while the volume of the corresponding unit-n-cube is 1.

Chapter 4

The multi-level *hp*-finite element method[1,2]



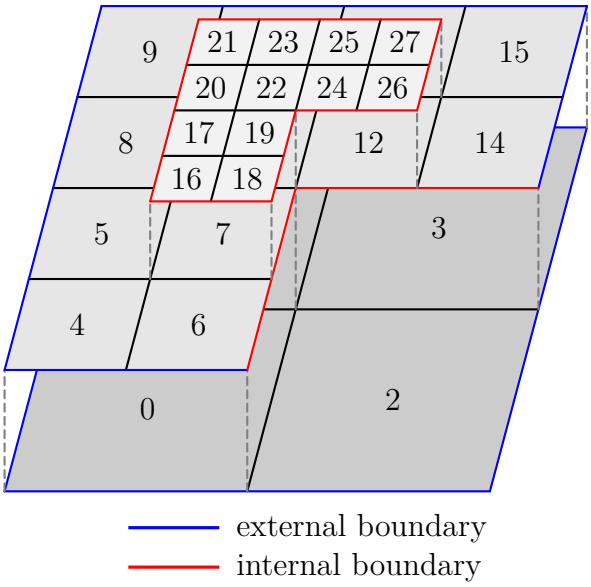
(a) Refine by replacement

(b) Refine by superposition

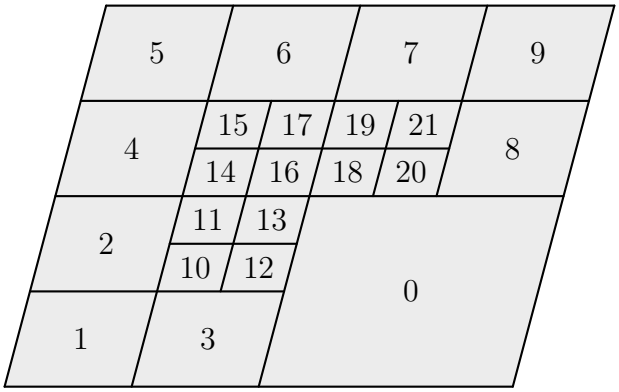
Figure 4.1: Comparison of *hp*-refinement strategies. From Kopp, Calo, et al. (2022), used under Creative Commons CC BY 4.0 license.

Refining the finite element discretization by only increasing the polynomial degree is often not the best strategy, as the element size must also be reduced to recover exponential convergence for solutions with reduced regularity. Even for smooth solutions, starting with several levels of h-refinement until the solution becomes smooth enough on each element is often more efficient, as p-refinement is more expensive per degree of freedom than h-refinement. However, replacing coarse finite elements with a set of finer ones introduces hanging nodes — element interfaces that are subdivided on one side but not on the other. The presence of hanging nodes greatly complicates the construction of continuous basis functions, as the shape functions of finer elements cannot be directly combined with shape functions on the coarser element. Instead, combinations of shape functions must be constrained and connected. The complexity of this process increases strongly with more spatial dimensions as well as mesh irregularity (Demkowicz et al., 1989; Demkowicz, 2006; Zander et al., 2016).

The multi-level *hp*-method circumvents the challenge of treating hanging nodes by keeping



(a) Full cell numbering



(b) Leaf cell (element) numbering

Cell	n_l	n_r	n_b	n_t	P	R	L
0	-1	2	-1	1	-1	0	0
1	-1	3	0	-1	-1	0	0
2	0	-1	-1	3	-1	0	1
3	1	-1	2	-1	-1	0	0
4	-1	6	-1	5	0	1	1
5	-1	7	4	8	0	1	1
6	4	2	-1	7	0	1	1
7	5	2	6	10	0	1	0
8	-1	10	5	9	1	1	1
9	-1	11	8	-1	1	1	1
10	8	12	7	11	1	1	0
11	9	13	10	-1	1	1	1
12	10	14	2	13	3	1	0
13	11	15	12	-1	3	1	1
14	12	-1	2	15	3	1	1
15	13	-1	14	-1	3	1	1
16	5	18	6	17	7	2	1
17	5	19	16	20	7	2	1
18	16	2	6	19	7	2	1
19	17	2	18	22	7	2	1
20	8	22	17	21	10	2	1
21	8	23	20	11	10	2	1
22	20	24	19	23	10	2	1
23	21	25	22	11	10	2	1
24	22	26	2	25	12	2	1
25	23	27	24	13	12	2	1
26	24	14	2	27	12	2	1
27	25	14	26	13	12	2	1

(c) Data structure

Figure 4.2: Essential topological data: left, right, bottom and top neighbors n_l, n_r, n_b, n_t, parent index P, refinement level R, and a flag L marking the leaf cells. From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

coarse elements during the refinement. Allowing these coarse elements to support shape functions in them naturally resolves cases where neighboring elements have a different refinement level. Instead of trying to connect the coarse shape functions to the finer ones, the functions on the original coarse neighbor (that is now overlayed with finer elements) are activated. The shape functions from neighboring elements are combined using this idea, essentially reducing the problem of constructing an *hp*-basis to creating multiple levels of *p*-finite element meshes. Figure 4.1 compares the replacement and overlay strategies and demonstrates how coarse shape functions are completed on the coarse parent of the finer neighbors.

4.1 Hierarchical refinement and data structure[1]

The refinement process starts with a base mesh, where cells marked for refinement are overlayed by 2^d subcells from bisecting the original cell in each coordinate direction. Performing this process recursively on overlay cells yields a 2^d-ary refinement tree. The choice to refine can use an error indicator or a priori knowledge of the solution. All cells within the refinement tree can support basis functions, although finite elements are chosen only as leaf cells.

A central aspect of the presented framework is to identify a lightweight data structure that can store all essential information for constructing an *hp*-basis. The algorithms introduced in this chapter require for each cell the index of its parent cell, its refinement level, the indices of its neighbors on the same or a lower refinement level, and the information whether it is a leaf. The neighbor indices are stored in a $(n_{\text{cell}}, d, 2)$ sized array (denoted as N), and for the other relations, simple one-dimensional arrays with one entry per cell are used. The roots of the refinement tree can be identified by choosing the parent cell index -1 for cells without a parent. An entry in the neighbor array can correspond to an *internal interface* (neighbor exists and has same refinement level), *internal boundary* (neighbor exists and has lower refinement level), and *external boundary* (neighbor does not exist, face is part of the domain boundary). External boundaries are assigned the *no-cell* value, and internal boundaries are assigned the index of the coarser leaf cell. Figure 4.2 sketches a simple example mesh with two refinement levels.

The multi-level *hp*-method introduces two main challenges: linear independence must be preserved and the overlay functions must be zero on internal boundaries to guarantee continuity in the hierarchical sum of all shape functions. The following section addresses these challenges and shows how to automatically resolve them with little implementation effort.

Remark 4.1.1

The internal boundaries are the multi-level hp equivalent of the hanging nodes from classical hp-finite elements. The treatment of hanging nodes introduces constraints to the shape functions on both sides. In contrast, the multi-level hp-method simply imposes homogeneous Dirichlet conditions on internal boundaries (by removing active shape functions on them).

Remark 4.1.2

One can exploit the redundancy in the data structure to reduce the amount of storage that is needed for the meshes. For example, the refinement levels and leaf flags can be trivially constructed from the parent indices on the fly. However, additional information, like the geometric mapping function of each cell, may also be needed.

4.2 Tensor-product masks[1]

This section constructs the tensor-product masks for all cells (leaves and non-leaves) in the refinement tree introduced in the previous section (with an arbitrary refinement depth). The tensor-product masks again activate the right shape functions on each cell, such that they can combine into globally continuous, linearly independent *hp*-basis functions that are complete up to the selected polynomial degrees inside each element. These polynomial degrees are only assigned to the leaves (the finite elements), and they can again vary in the local directions.

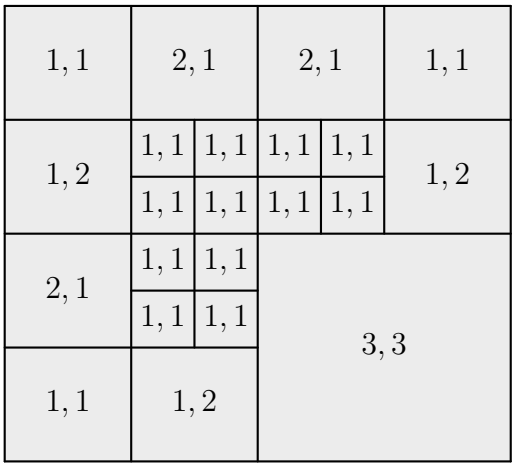


Figure 4.3: Polynomial degrees for each finite element (Figure 4.2b) and coordinate direction. From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

Figure 4.3 shows the polynomial degrees chosen for the example mesh introduced in Figure 4.2. The following algorithm extends the ideas of Section 3.3 to construct the multi-level *hp* tensor-product masks in four steps (see Figures 4.4 - 4.8 and Algorithm 3).

1. Initialize leaf masks according to their polynomial degree and non-leaf masks empty. Activating the complete tensor-product or trunk space only on the leaves naturally yields shape functions that are linearly independent over the hierarchy, as none of the parents contribute any shape functions. Choosing the trunk space masks derived in Section 3.5 instead of activating the full tensor-product does not require any changes in the remaining parts of the algorithm. Figure 4.4 shows the initial tensor-product masks with the corresponding shape functions using the full tensor-product space according to the degrees specified by Figure 4.3. In the current state, the shape functions cannot be combined into continuous basis functions yet as they are not compatible along the cell interfaces (red color). Moreover, the shape functions on the internal boundaries on the first and second overlay meshes are still active, as no Dirichlet conditions were imposed yet (violet color).

2. Restore interface compatibility using logical *or* operations in d iterations. In a loop over the internal interfaces, a tensor-product mask entry is activated if the corresponding entry in the neighbor is active. Internal interfaces bound two cells of the same refinement level, which makes this step equivalent to treating the cells of each refinement level as separate p-finite element meshes and applying the algorithm of Section 3.3 to them independently. This serves two purposes: first, it resolves incompatible polynomial degrees by the *maximum degree strategy*, and second, it activates the interface shape functions on the parent elements of finer neighbors. Figures 4.5 - 4.6 demonstrate the two necessary iterations in the current step of the tensor-product mask creation for the two-dimensional example introduced earlier. The shape functions on the bottom right cell of level zero can now be completed on the neighboring cells of the same level, although these cells have been refined and now also support up to two levels of finer overlay cells. Two iterations of this algorithm activate all four linear shape functions that together form the central hat basis function on level zero. The second iteration activates, in this example, the single bilinear shape function in cell 1 (top left) on level zero.

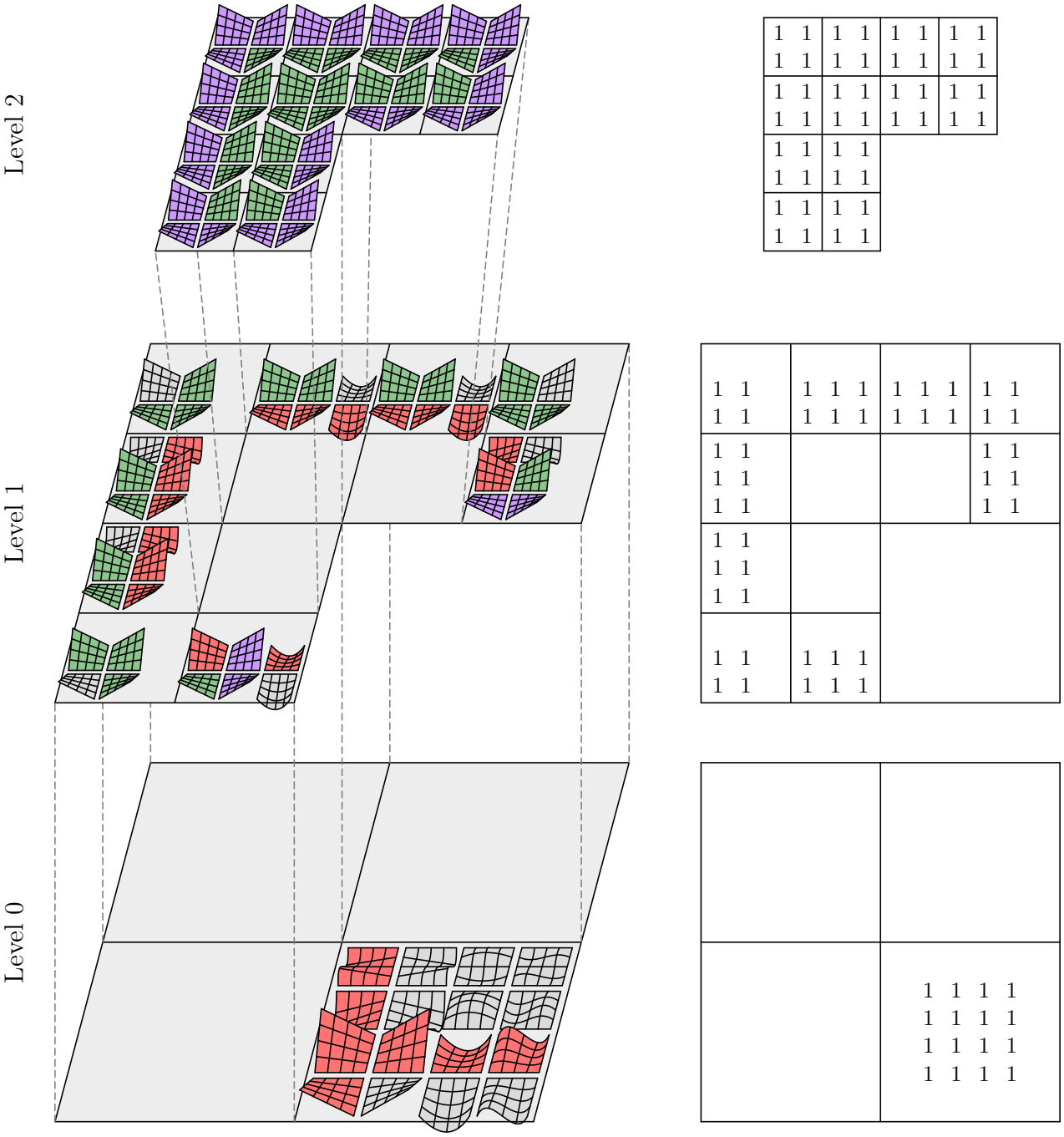


Figure 4.4: Initial tensor-product masks corresponding to degrees in Figure 4.3. From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

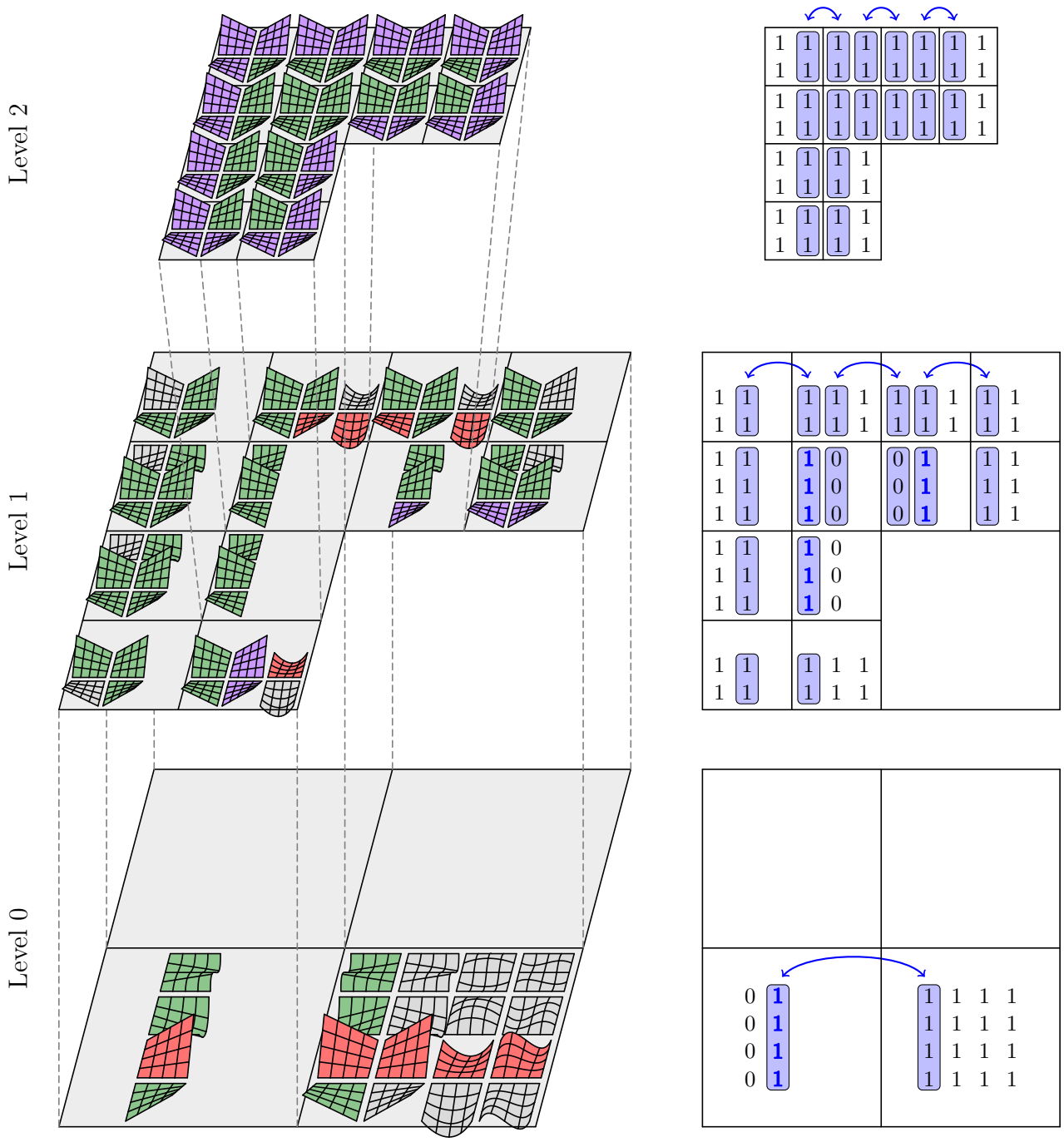


Figure 4.5: Restore continuity after initial activation: Slices normal to x-axis. From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

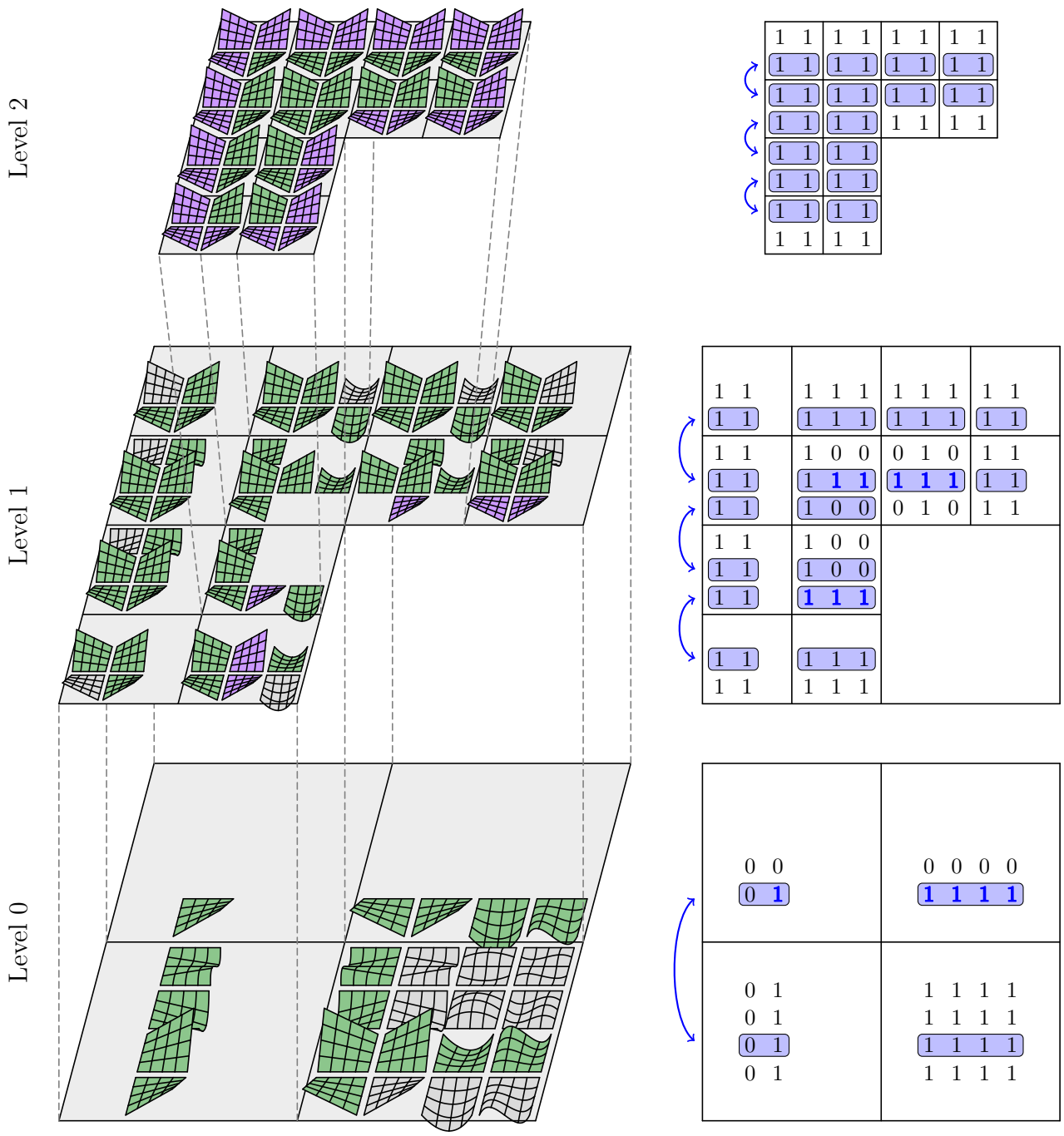


Figure 4.6: Restore continuity after initial activation: Slices normal to y-axis. From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

Figure 4.7: Deactivate internal boundaries (step 3). From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

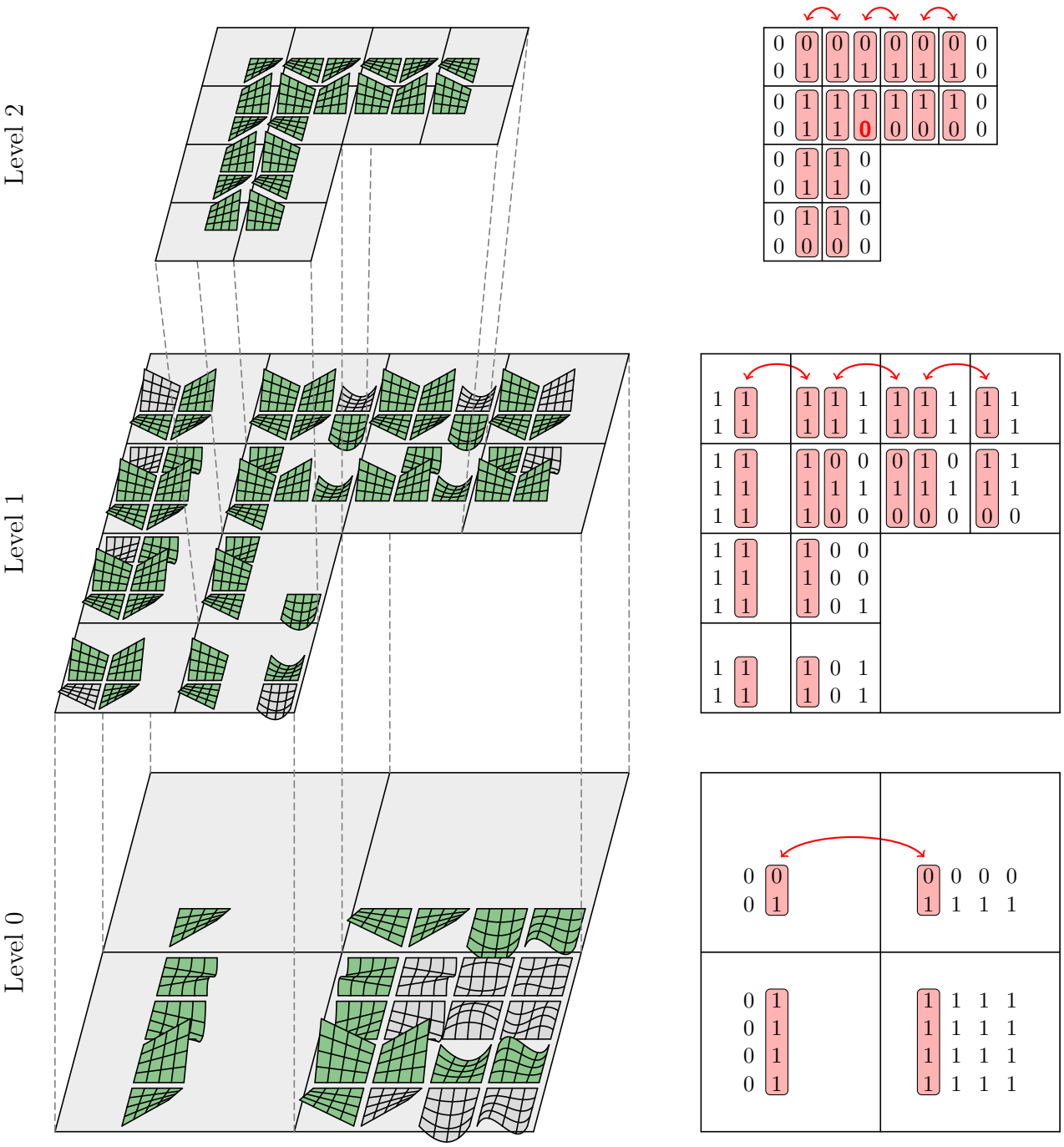


Figure 4.8: Communicate zero Dirichlet conditions to neighbors (step 4). From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

Algorithm 3 Construct multi-level *hp* tensor-product masks.

```
// Replaces p-fem version; requires neighbor to exist with same level
void operateOnInterfaces(NdArrayList<Type>& A, Neighbors N, Levels L, Op op)
{
    for(int a from 0 to d - 1)              // For d coordinate axes
        for(int i from 0 to size(A) - 1)    // For all cells
            if(int N_a = N(i, a, 1); N_a ≠ -1 and L(i) == L(N_a))
                operateOnInterface(A(i), A(N_a), a, op);
}
// Deactivate first (f == 0) or second (f == 1) array slice normal to axis n
void deactivateSlice(Mask& M_i, int n, int f)
{
    Vector s^n = removeEntry(sizes(M_i), n);

    for(Vector i^n in productIndices(s^n))
        M_i(insertEntry(i^n, n, f)) = false;
}
// Deactivates face functions if neighbor exists with different level
void deactivateOnInternalBoundaries(MaskList& M, Neighbors N, Levels L)
{
    for(int a from 0 to d - 1)              // For d coordinate axes
        for(int f from 0 to 1)              // For two faces
            for(int i from 0 to size(M))    // For all cells
                if(int N_a = N(i, a, 1); N_a ≠ -1 and L(i) != L(N_a))
                    deactivateSlice(M(i), a, f);
}

MaskList createMlhpMasks(Neighbors N, Levels L, Isleaf E, Degrees p)
{
    MaskList M; int l = 0;

    // Step 1: Initialize leaf masks (could use trunk space here instead)
    for(bool i from 0 to size(E) - 1)
        M.append(Mask(p(l++) + 1, true) if E(i) else Mask(0));

    // Step 2: Restore compatibility by activating on neighbors
    for(int it from 0 to d - 1)
        operateOnInterfaces(M, N, L, logicalOr);

    // Step 3: Impose Dirichlet on internal boundaries
    deactivateOnInternalBoundaries(M, N, L);

    // Step 4: Restore compatibility by deactivating on neighbors
    for(int it from 0 to d - 2)
        operateOnInterfaces(M, N, L, logicalAnd);

    return M;
}

void createMlhpLocationMatrices(MaskList M, Neighbors N, Levels L)
{
    LocationMatrixList G, int n_ids = initializeGlobalIndices(M);

    for(int it from 0 to d - 1)
      operateOnInterfaces(G, N, L, maximum);

    removeUnassignedIndices(G, n_ids);

    return G;
}
```

Algorithm 4 Evaluate multi-level *hp*-basis functions.

```
// Evaluate multi-level hp basis at local coordinates r of cell i, differentiated
// k(a) times in direction a. For example, k = (1, 0) evaluates ∂N_j/∂r_0 in 2D and
// k = (0, 1, 1) evaluates ∂²N_j/(∂y∂r_2) in 3D. Note that this function only maps
// the derivatives into the leaf coordinate system, not yet into global space.
Vector evaluate(Mesh mesh, MaskList M, int i_l, Coords r, Diff k)
{
    Vector N = [ ];   // shape functions
    double n_L = 0;    // number of levels

    // Loop from leaf to root: start with full index of leaf and update to
    // parent in each new iteration until current cell has no parent (-1)
    for(int i = i_l; i ≠ -1; i = mesh.parent(i))
    {
        Mask M_i = M(i);
        Vector s = sizes(M_i);

        if(product(s) ≠ 0)
        {
            // Evaluate integrated Legendre polynomials
            VectorList I;

            for(int a from 0 to d - 1)
                I.append(integratedLegendre(r(a), s(a) - 1, k(a)));

            // Append active entries in tensor-product to N
            for(Vector α in productIndices(s))
            {
                if(M_i(α))
                {
                    // Map derivatives to leaf
                    double N_α = power(1/2, n_L * sum(k));

                    // Multiply in directions j with polynomial at index α(j),
                    // differentiated k(j) times, evaluated at r(j)
                    for(int j from 0 to d - 1)
                        N_α = N_α * I(j)(α(j))

                    N.append(N_α);
                }
            }
        }

        // Map coordinate to parent
        for(int a from 0 to d - 1)
        {
            int N_a = mesh.neighbor(i, a, 0);

            double c = -1 if N_a == -1 or mesh.parent(N_a) != mesh.parent(i) else 1

            r(a) = (r(a) + c) / 2;
        }

        n_L = n_L + 1;
    }

    return N;
}
```

3. Deactivate shape functions on internal boundaries. The final step of constructing a multi-level *hp*-basis is to filter basis functions that are non-zero on the internal boundaries of the overlays (i.e., homogeneous Dirichlet conditions must be imposed). Thus, entries in the tensor-product masks are deactivated for shape functions that are non-zero on faces with a coarser neighbor. As Figure 4.7 shows, this step deactivates the shape functions on levels one and two that are active on the internal boundary to the bottom right cell of level zero. Note that the shape functions deactivated here are already active on a parent cell due to the preceding compatibility recovery in the second step. Moreover, external boundaries are not modified as there are no hanging nodes, and the finest resolution is desirable there (as already initialized in step 1).

4. Restore interface compatibility using logical *and* operations in $d-1$ iterations. The basis functions on internal interfaces often receive contributions from additional elements. For example, on level two of Figure 4.7, the basis function associated with the node in the central corner is built from three linear shape functions supported on the three surrounding elements (cells 19, 22, and 24), but only two of them are deactivated (on cells 19 and 24, as only they are directly connected to the internal boundary). Such remaining contributions are deactivated by repeating the second step of the algorithm, but with one less repetition and with logical *and* operations. This deactivates interface shape functions if they are inactive in the neighbor, which only affects the changes from step 3, as all other interfaces are compatible after step 2. Figure 4.8 shows the final step of this algorithm for creating compatible multi-level *hp* tensor-product masks.

The tensor-product masks allow an evaluation of the basis functions in each finite element by first evaluating the integrated Legendre polynomials in each direction and then computing the active entries within the tensor-product according to the corresponding mask. The derivatives are mapped into the global coordinate system (physical space) if needed. The evaluation starts on the leaves and is repeated for each parent cell in the hierarchy towards the root cell. With each new level, the evaluation coordinates must be mapped accordingly. Algorithm 4 shows the basis function evaluation for a multi-level *hp*-element.

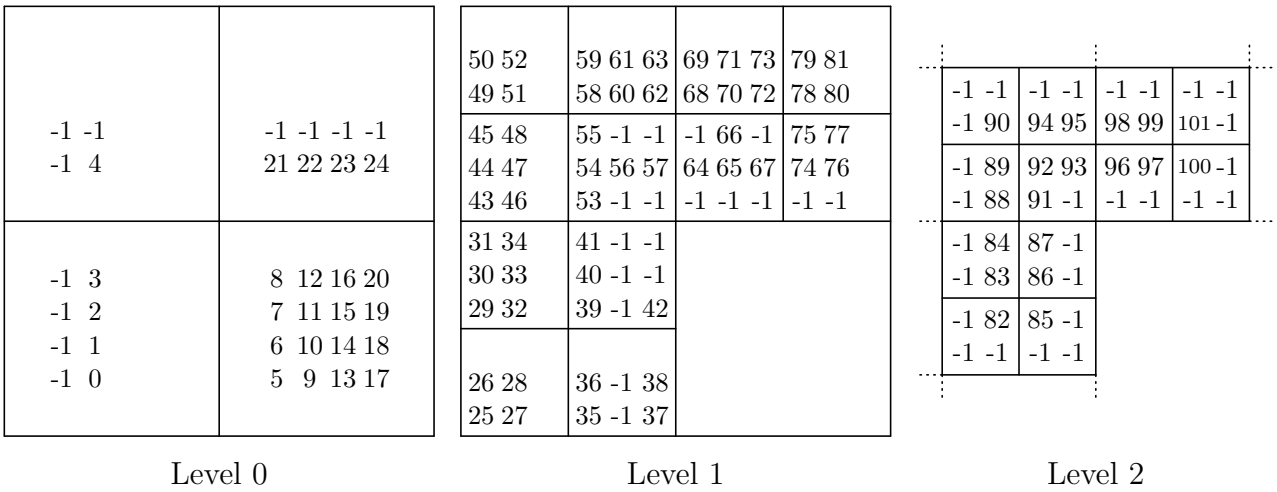


Figure 4.9: Initialize global ids independently. From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

4.3 Location matrices[1]

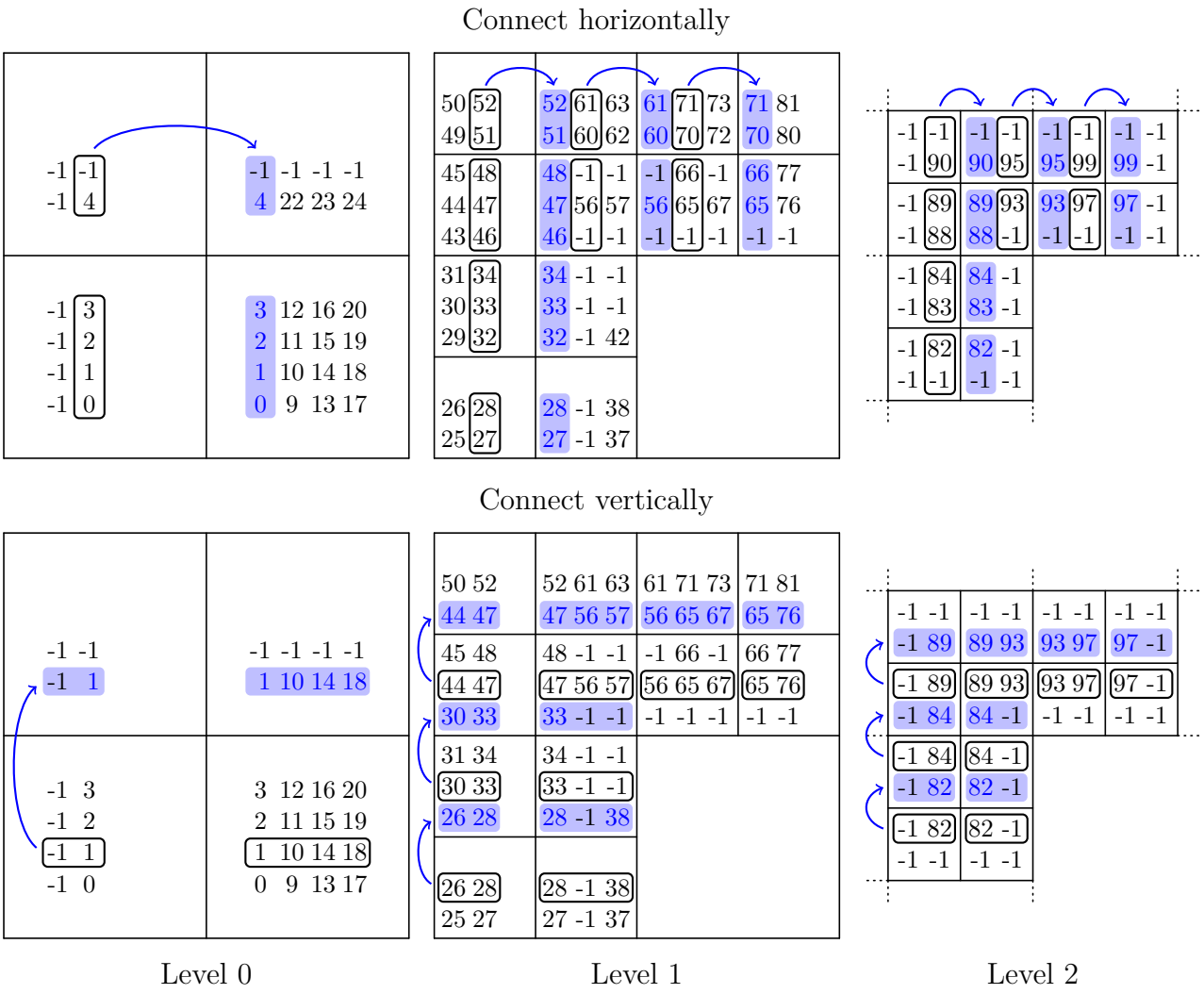


Figure 4.10: Connect global ids over cell interfaces. From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

In this section, the active shape functions on the cells of the hierarchical refinement tree from the previous section are connected to global multi-level *hp*-basis functions by using the ideas of Section 3.3. Each refinement level is again treated like a separate *p*-finite element mesh to construct the location matrices using the same steps as in Section 3.3. In other words: the shape functions are connected only across internal interface and not across interfaces to coarser neighbors. In contrast to the multi-level *hp* tensor-product masks, the construction of the location matrices does not need to be further adapted to account for the mesh hierarchy and the different types of cell faces. Figure 4.9 shows the initial location matrices corresponding to the tensor-product masks from Figure 4.8, where each shape function is assigned a unique index. These are then connected in both coordinate directions in Figure 4.10, and the resulting gaps in the global numbering are eliminated in Figure 4.11 by applying an index map like in Figure 3.11d. The complete set of multi-level *hp* basis functions is shown in Figures 4.12 - 4.13.

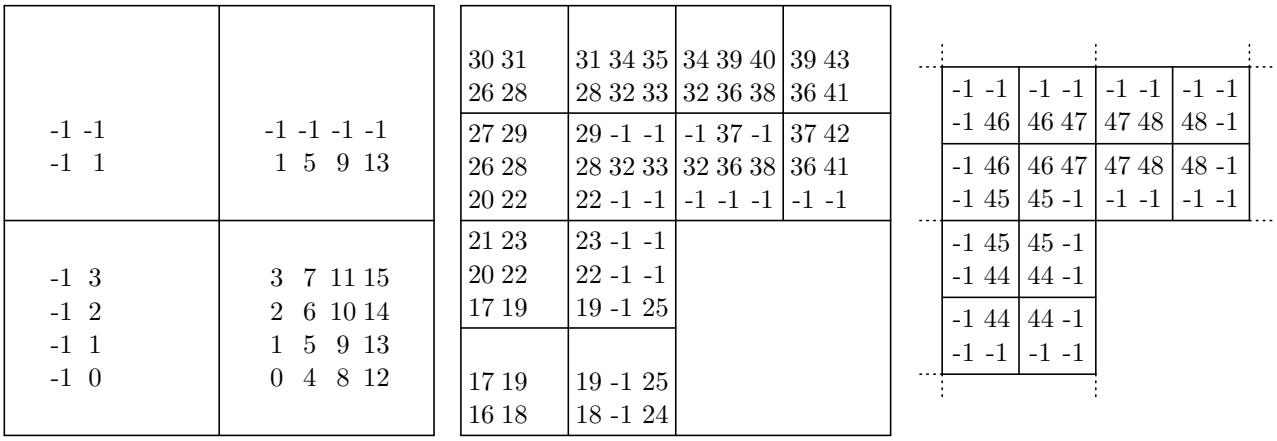


Figure 4.11: Eliminate unassigned global basis function indices. From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

The last function of Algorithm 3 constructs the location matrices for a multi-level *hp*-basis.

Remark 4.3.1

The example features only two levels of refinement. The algorithms, however, apply to arbitrary nested spacetree partitions. They only distinguish between leaf- and non-leaf cells and between different types of cell faces.

Remark 4.3.2

In practice, one might encapsulate the topological and geometric data from Section 4.1 into a mesh data type and the results of Sections 4.2 - 4.3 into an hp-basis data type.

4.4 Simulation workflow[1]

In a computation with multi-level *hp*-finite elements, the algorithms from Sections 4.2 and 4.3 directly follow the mesh creation to prepare the tensor-product masks and location matrices. Both are defined on all cells of the refinement tree. Theoretically, one could consider each cell a finite element and assemble their contributions separately. However, the volumetric coupling of the basis functions through the hierarchy renders this approach impractical, as the interactions between all the cells in the tree must be integrated. Moreover, the concept of overlapping finite elements is not very transparent for someone concerned with practical applications who may not be familiar with the details of the method. Instead, only the leaves are considered finite elements. When evaluating the shape functions, the contributions of all parent elements are simply appended while traversing the hierarchy and mapping the evaluation coordinates accordingly. Therefore, from ”outside”, the multi-level *hp*-basis consists of the non-overlapping leaves, where some of them evaluate more shape functions than others (e.g., in transition zones with many hanging nodes). Hiding the hierarchical nature of the basis (which still exists internally) allows treating the multi-level hp-basis like any other finite element method and applying standard algorithms. Equivalent to gathering the shape function over the full

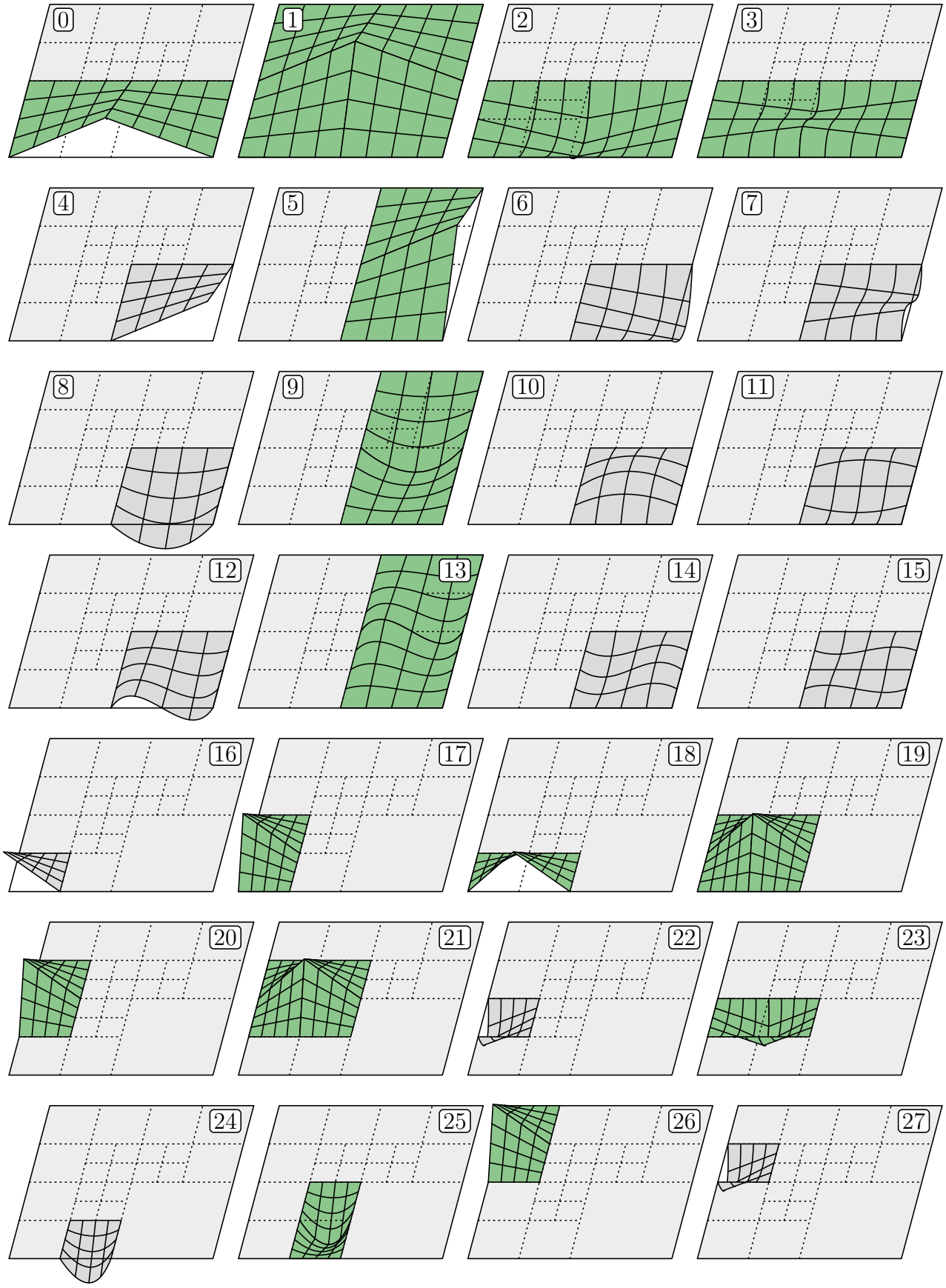


Figure 4.12: Multi-level *hp* basis functions 0 to 27 according to Figure 4.11.

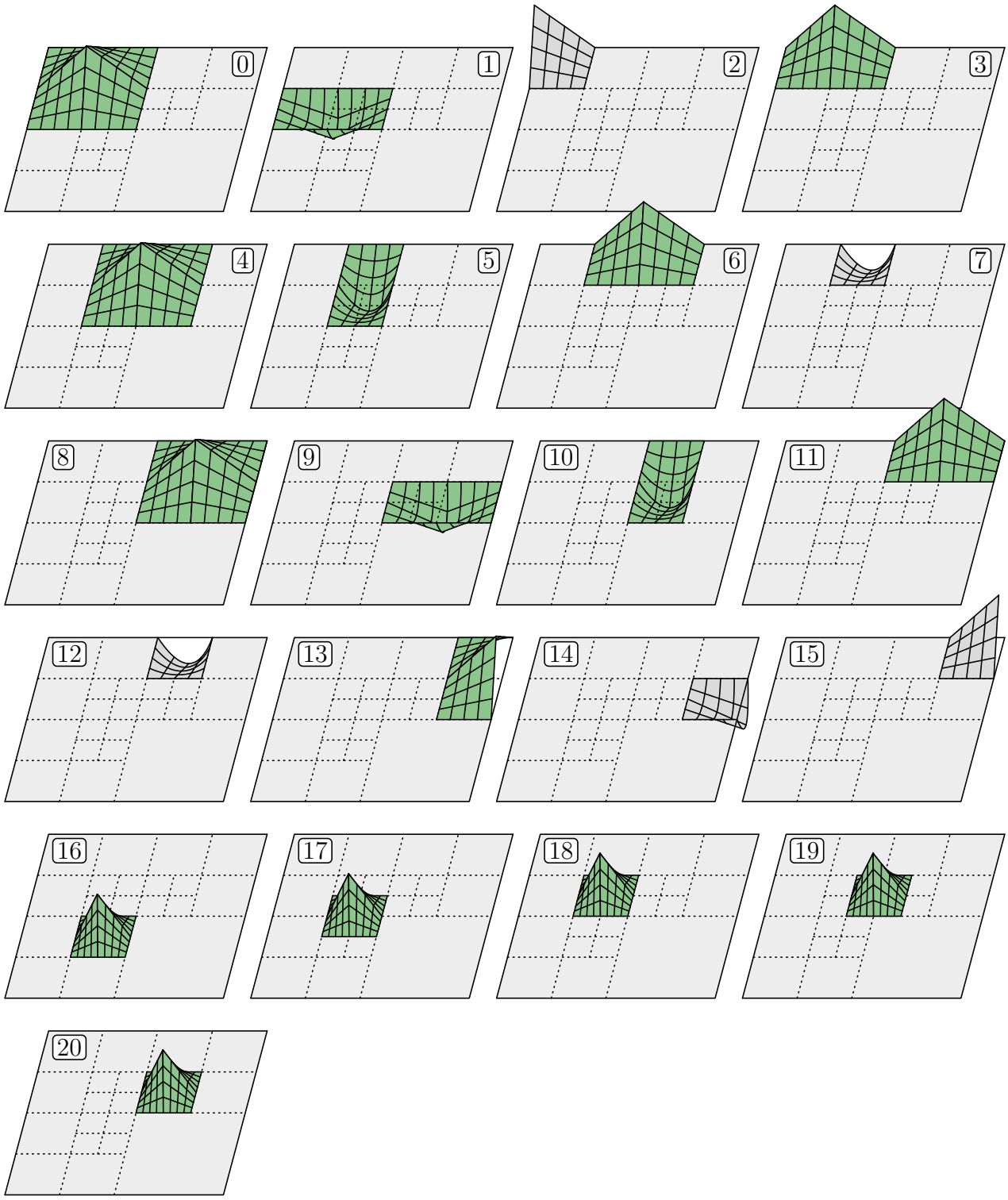


Figure 4.13: Multi-level *hp* basis functions 28 to 48 according to Figure 4.11.

element 0 : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
element 1 : [16, 17, 18, 19, 0, 1, 2, 3]
element 2 : [17, 20, 21, 19, 22, 23, 0, 1, 2, 3]
element 3 : [18, 19, 24, 25, 0, 1, 2, 3]
element 4 : [20, 26, 27, 22, 28, 29, 1]
element 5 : [26, 30, 28, 31, 1]
element 6 : [28, 31, 32, 34, 33, 35, 1]
element 7 : [32, 34, 36, 39, 38, 40, 1, 5, 9, 13]
element 8 : [36, 37, 41, 42, 1, 5, 9, 13]
element 9 : [36, 39, 41, 43, 1, 5, 9, 13]
element 10 : [44, 19, 22, 23, 25, 0, 1, 2, 3]
element 11 : [44, 45, 19, 22, 23, 25, 0, 1, 2, 3]
element 12 : [44, 19, 22, 23, 25, 0, 1, 2, 3]
element 13 : [44, 45, 19, 22, 23, 25, 0, 1, 2, 3]
element 14 : [45, 46, 22, 28, 29, 32, 33, 1]
element 15 : [46, 22, 28, 29, 32, 33, 1]
element 16 : [45, 46, 47, 22, 28, 29, 32, 33, 1]
element 17 : [46, 47, 22, 28, 29, 32, 33, 1]
element 18 : [47, 48, 32, 36, 37, 38, 1, 5, 9, 13]
element 19 : [47, 48, 32, 36, 37, 38, 1, 5, 9, 13]
element 20 : [48, 32, 36, 37, 38, 1, 5, 9, 13]
element 21 : [48, 32, 36, 37, 38, 1, 5, 9, 13]

Figure 4.14: Element location maps including parent cells (colors represent parent levels). From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

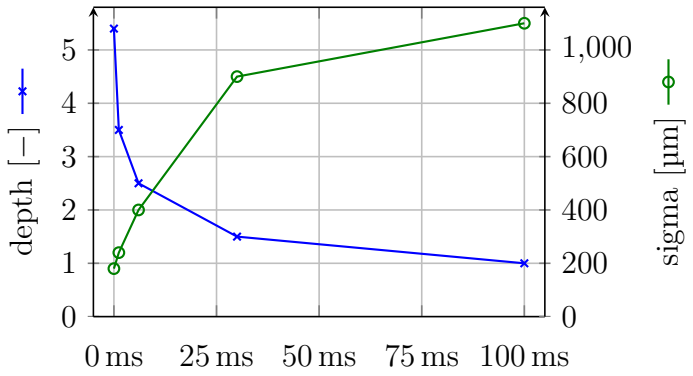
hierarchy, the global basis function indices in all location matrices are concatenated into a single location map for each finite element. Figure 4.14 shows these location maps for the hierarchical example of this chapter. As many finite elements share the same parent shape functions, some indices appear quite often. This redundancy is eliminated by storing the location maps for each cell and only combine them on the fly for individual leaves when necessary.

After the mesh and basis construction, the location maps of the multi-level *hp*-basis are used to allocate a sparse matrix in a compressed-sparse-row (CSR) format and a dense load vector. This ensures that the subsequent assembly loop over all elements can add the element matrices to the global matrix without allocating new non-zero entries. For each element, the corresponding location map is used to allocate a dense element matrix and an element load vector according to the length of the location map. The number of Gauss-Legendre points per direction is determined by querying the basis for the maximum polynomial degrees of the element, which are obtained from the sizes of the tensor-product masks of the element and its parents. For each integration point (usually $p+1$ per direction), the shape functions and their derivatives are evaluated, and the finite element integrals are added to the element system. Finally, the element matrix and element load vector are assembled into the global system using the current location map.

The resulting global linear system is then solved using either a diagonally preconditioned conjugate gradient method (time-stepping) or the Pardiso sparse direct solver (space-time slab). Both perform well for any number of refinement levels on the examples shown in this thesis. The multi-level *hp*-method has been combined with Additive-Schwarz (Jomo et al., 2018) and multigrid preconditioners (Jomo et al., 2021) (exploiting the hierarchical nature) or with sparse direct solvers (Elhaddad et al., 2017). After the linear solution, quantities of interest, like the solution field, are written to the hard drive using the VTU file format for visualization in Paraview. The high-order nature of the approximation is captured by

evaluating the solution field inside an element on a finer sample grid whose resolution is chosen in relation to the polynomial degree.

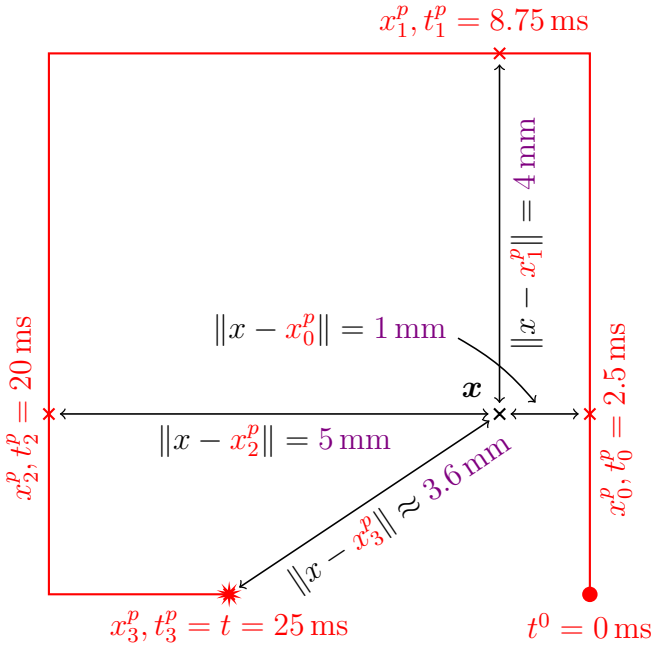
4.5 Refinement strategy based on laser path[2]



delay [ms]	depth [−]	sigma [µm]
0.0	5.4	180
1.2	3.5	240
6.0	2.5	400
30.0	1.5	900
100.0	1.0	1100

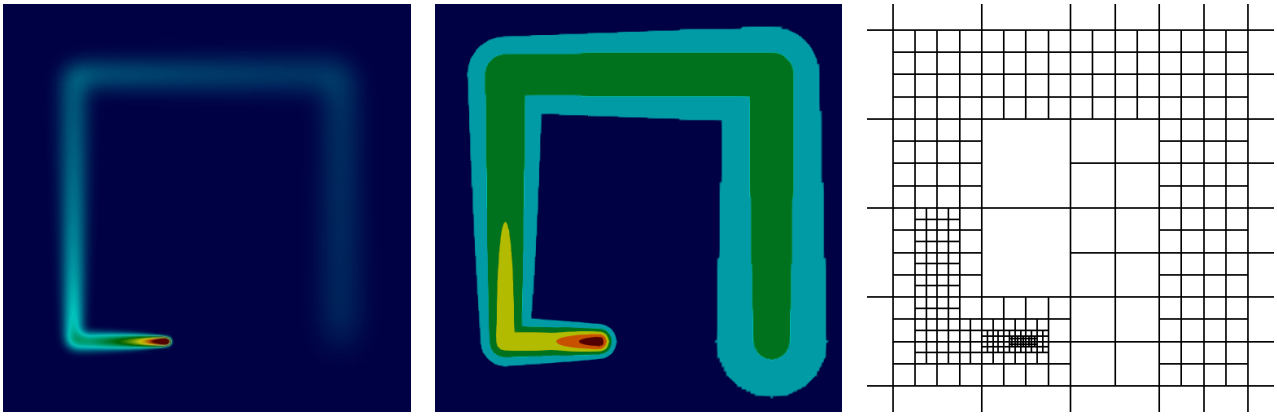
Figure 4.15: Parameters for refinement based on laser history inside the refinement window $[t-\tau_{\max}, t]$. From Kopp, Calo, et al. (2022), used under Creative Commons CC BY 4.0 license.

It is crucial to find strategies for automatic mesh refinement, mesh coarsening, and polynomial degree selection to use the flexibility offered by an *hp* framework to its full potential. General approaches use error estimators combined with some smoothness estimators to choose whether to refine in h or p. While these require little problem setup information, they are often expensive and complex to implement. However, in applications such as PBF-LB/M, the a



$$
\begin{aligned}
d_0 &= d_\tau\,(22.5\,\text{ms}) \cdot \exp\left(-\tfrac{(1\,\text{mm})^2}{2\sigma_\tau^2(22.5\,\text{ms})}\right) \\
&\approx 1.81 \cdot \exp\left(-0.5 \cdot 1^2 \cdot 0.744^{-2}\right) \approx 0.73 \\
d_1 &= d_\tau\,(16.25\,\text{ms}) \cdot \exp\left(-\tfrac{(4\,\text{mm})^2}{2\sigma_\tau^2(16.25\,\text{ms})}\right) \\
&\approx 2.07 \cdot \exp\left(-0.5 \cdot 4^2 \cdot 0.613^{-2}\right) \approx 0 \\
d_2 &= d_\tau\,(5\,\text{ms}) \cdot \exp\left(-\tfrac{(5\,\text{mm})^2}{2\sigma_\tau^2(5\,\text{ms})}\right) \\
&\approx 2.71 \cdot \exp\left(-0.5 \cdot 5^2 \cdot 0.367^{-2}\right) \approx 0 \\
d_3 &= d_\tau\,(0\,\text{ms}) \cdot \exp\left(-\tfrac{(3.6\,\text{mm})^2}{2\sigma_\tau^2(0\,\text{ms})}\right) \\
&\approx 5.4 \cdot \exp\left(-0.5 \cdot 3.6^2 \cdot 0.180^{-2}\right) \approx 0 \\
d &= \max\{d_0, d_1, d_2, d_3\} = 0.73 \to 1
\end{aligned}
$$

Figure 4.16: Evaluation of the depth function $d(x,t)$ at $x = (4\,\text{mm}, 1\,\text{mm})$, $t = 25\,\text{ms}$ by finding the closest point on each laser path segment (a square contour with 6 mm side length).



(a) Computed temperature (b) Refinement depth function (c) Finite element mesh

Figure 4.17: The depth function in the middle defines the target number of multi-level hp refinements. It evaluates (4.1) using the path shown in Figure 4.16 and the parameters from Figure 4.15. The resulting mesh to the right captures the temperature on the left side well. From Kopp, Calo, et al. (2022), used under Creative Commons CC BY 4.0 license.

priori knowledge about the laser path can be used to tailor mesh refinement strategies. In this thesis, the mesh is refined by considering recent laser positions, called the refinement window (e.g., from 100 ms in the past until the present time), for constructing a function conceptually similar to a mesh density function that defines the target refinement depth. This function is constructed starting with a Gauss bell shape at the current position of the laser with a maximum refinement depth at the center that decreases while moving further away from the laser. Then, this function is transported in space along the previous laser path, decreasing the maximum refinement level and increasing the refinement width. This transport yields a spatial function for a given point in time with values indicating the target refinement level. The decision to refine an element is made by evaluating the refinement depth function on a grid of points within the element (e.g., 5 to 7 per direction) and comparing the maximum target refinement depth to the refinement level of the element. Figure 4.15 shows the maximum refinement depth d_τ and width σ_τ defined over the refinement window (the recent laser history) as used later in the examples.

The refinement depth function $d(x,t)$ is evaluated by considering the laser path in the time interval $[t-\tau_{\max}, t]$; the regions before and after do not influence the refinement. The function evaluation first computes the closest point x_i^p on each laser path segment i and determines the time t_i^p at which the laser was at x_i^p. Then, using the time delay $\Delta t_i = t - t_i^p$, the maximum refinement depth and width are extracted from the functions $d_\tau(\Delta t)$ and $\sigma_\tau(\Delta t)$, respectively; see Figure 4.15. These allow computing $d(x,t)$ by evaluating a Gaussian function for each segment and taking the maximum value:

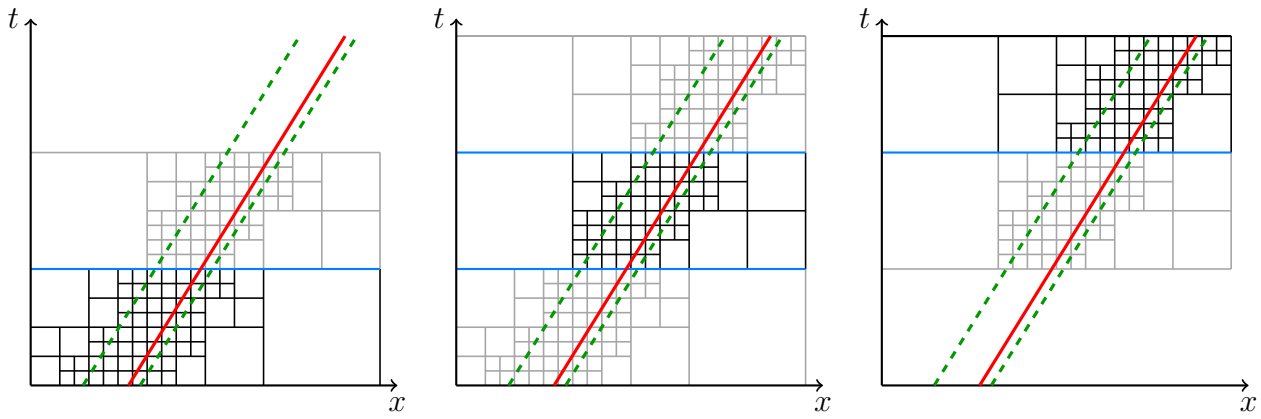
$$d(x,t) = \max_i d_i(x,t) = \max_i \left(d_\tau\left(t - t_i^p\right) \exp\left(-\frac{\|x - x_i^p\|^2}{2\sigma_\tau^2(t - t_i^p)} \right) \right), \tag{4.1}$$

rounding the result to the closest integer. Figure 4.16 exemplifies this process for a square contour path with four segments. The right segment dictates a maximum of one level of refinement after rounding due to its proximity to the evaluation point. All other segments are too far away in relation to the width of the Gauss function, interpolated from Figure 4.15.

Figure 4.17 shows the evaluation of $d(x,t)$ on the entire two-dimensional domain, the resulting mesh, and the computed temperature field. In this example, the refinement window in Figure 4.15 acts inside the interval $[t-100\,\text{ms}, t]$. Hence, all four segments must be considered, as the total printing time does not exceed 100 ms. However, for longer laser paths (or differently defined refinements), the segments whose time interval does not intersect the window of active refinement ($[t-\tau_{\max}, t]$) may be discarded for a given time t.

As the presented multi-level *hp*-refinement is isotropic, the same refinement depth applies to all directions, including time. For simplicity, the same polynomial degree is chosen for elements with equal refinement levels, which is not a restriction of the method. Finding better selection criteria for the polynomial degrees of individual elements will improve the simulation performance; future work will focus on this.

4.6 Slab compatibility[2]



(a) First slab (b) Second slab (c) Third and last slab

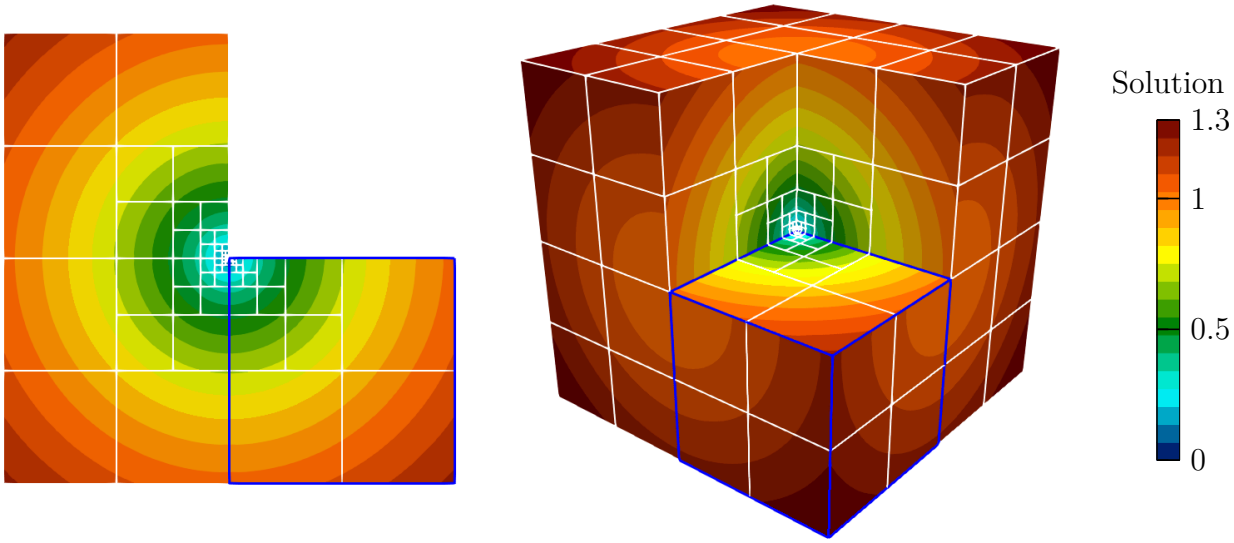
Figure 4.18: Ghost slabs in time with one space dimension. From Kopp, Calo, et al. (2022), used under Creative Commons CC BY 4.0 license.

Another challenge in separating the solution into time slabs is that hanging nodes are not constrained when meshing each slab separately. Meshing the entire space-time domain at once is certainly not desirable. Still, the refinement for the next and previous slabs must be known to construct compatible bases at slab interfaces. Figure 4.18 shows how a one-dimensional solution is advanced in time by meshing the second slab together with the first one. This allows the basis of the first slab to "know" the hanging nodes on the time interfaces to the second slab. Then, before creating the basis for the second slab, the third one is meshed, and only when computing the solution on the third slab is the first mesh discarded. This meshing procedure is possible as the refinement does not depend on the solution and is defined in advance. In other cases (e.g., when using an adaptive algorithm), one can either refine the next slab conformingly or use a DG formulation in time to enforce continuity across slab interfaces weakly (see, e.g., Hofer et al., 2018; Schmich & Vexler, 2008).

Chapter 5

Results[1,2]

5.1 Singular benchmark[1]



(a) L-shaped domain (b) Fichera corner

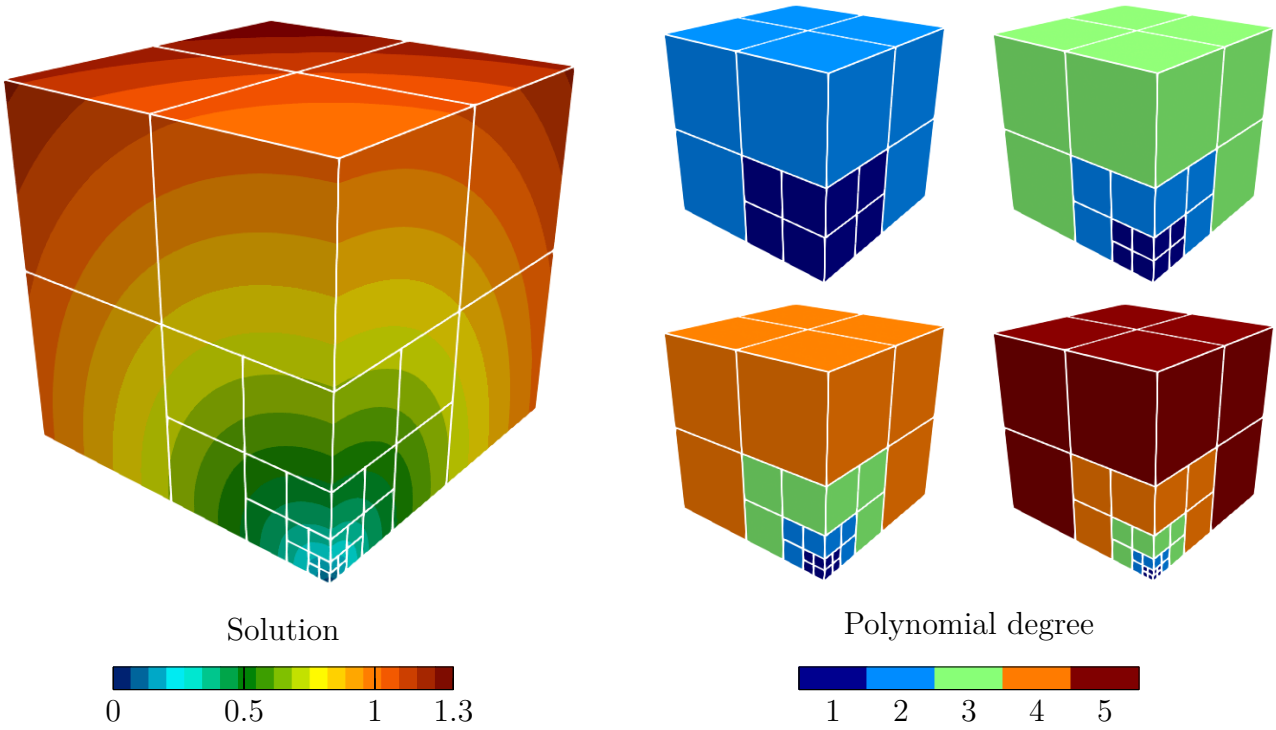
Figure 5.1: Singular benchmark solutions. From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

In this first example, the presented multi-level *hp*-algorithms and their implementation are verified using a linear Poisson problem

$$\begin{aligned} \nabla \cdot (\kappa \nabla u) &= f \qquad \text{on } \Omega \\ u &= g \qquad \text{on } \Gamma_D \\ \kappa \nabla u \cdot n &= h \qquad \text{on } \Gamma_N, \end{aligned} \tag{5.1}$$

where $\Gamma_D \cup \Gamma_N = \partial\Omega$ and $\Gamma_D \cap \Gamma_N = \emptyset$. The weak form of (5.1) reads as follows: Find $u \in u_g + H_0^1(\Omega)$, such that

$$\int_\Omega \nabla w \cdot \kappa \nabla u \, \mathrm{d}\Omega = \int_\Omega w f \, \mathrm{d}\Omega + \int_{\Gamma_N} w h \, \mathrm{d}\Gamma_N \qquad \forall w \in H_0^1(\Omega).$$



(a) Finite element solution

(b) Graded polynomial degrees for $r = 1$ to $r = 4$

Figure 5.2: Singular benchmark on one octant with four refinement levels. From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

The manufactured solution is defined as follows:

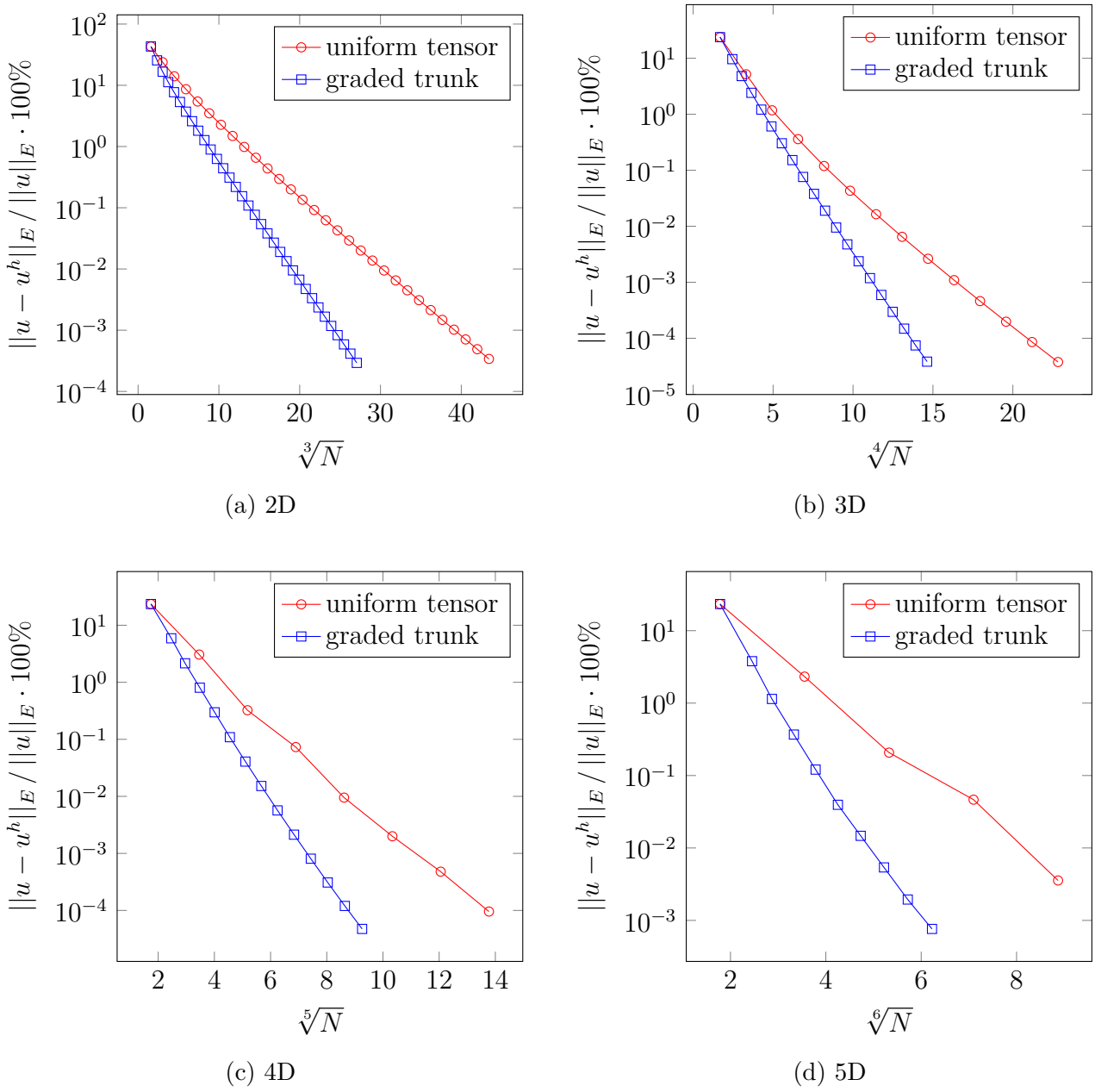
$$u = \sqrt{r} \qquad \text{with} \qquad r = \sqrt{\sum_{i=0}^{d-1} x_i^2} \tag{5.2}$$

on $\Omega = [0, 1]^d$ with $d \geq 2$, see Figure 5.2a. Substituting (5.2) into (5.1) and using $\kappa = 1$ yields

$$f = \frac{3 - 2d}{4} r^{-3/2}.$$

Homogeneous Neumann boundary conditions are imposed on the boundaries intersecting the coordinate planes: the left and bottom edges in two dimensions and the left, bottom, and front faces in three dimensions. Dirichlet boundary conditions are imposed on the remaining faces according to (5.2). The finite element discretization of (5.1) results in a linear equation system $\mathbf{K}\hat{\boldsymbol{u}} = \boldsymbol{F}$. In two dimensions, this benchmark is equivalent to computing one quadrant of an L-shaped domain and exploiting the symmetry of the solution (Figure 5.1a). Similarly, the three-dimensional analog is computing one octant of a Fichera cube with a corner singularity (Figure 5.1b).

To obtain exponential convergence, the finite element mesh must be refined locally towards the point singularity while elevating the polynomial degree elsewhere to approximate the smooth solution appropriately. The domain $\Omega = [0, 1]^d$ is discretized using a base mesh with two elements in each direction. Two refinement studies are performed, where each new computation adds another refinement level towards the singularity at the origin. The first



(a) 2D (b) 3D

(c) 4D (d) 5D

Figure 5.3: Convergence of the error in the energy norm. From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

strategy uses a full tensor-product space with a uniform polynomial degree p equal to $r + 1$, where r is the maximum refinement level. The second strategy employs a trunk space with a linear grading from $p = r + 1$ on the base elements to $p = 1$ on the elements with refinement level r. Figure 5.2b shows the graded polynomial degree distribution with one to four refinement levels and the numerical solution using a uniform tensor-product space with $p = 5$ and $r = 4$. Figure 5.3 shows the convergence results for different spatial dimensions.

While the convergence of the discretization error versus the number of unknowns is necessary to gain insight into the method's approximation quality, the method's success ultimately depends almost exclusively on runtime performance and memory consumption. The following

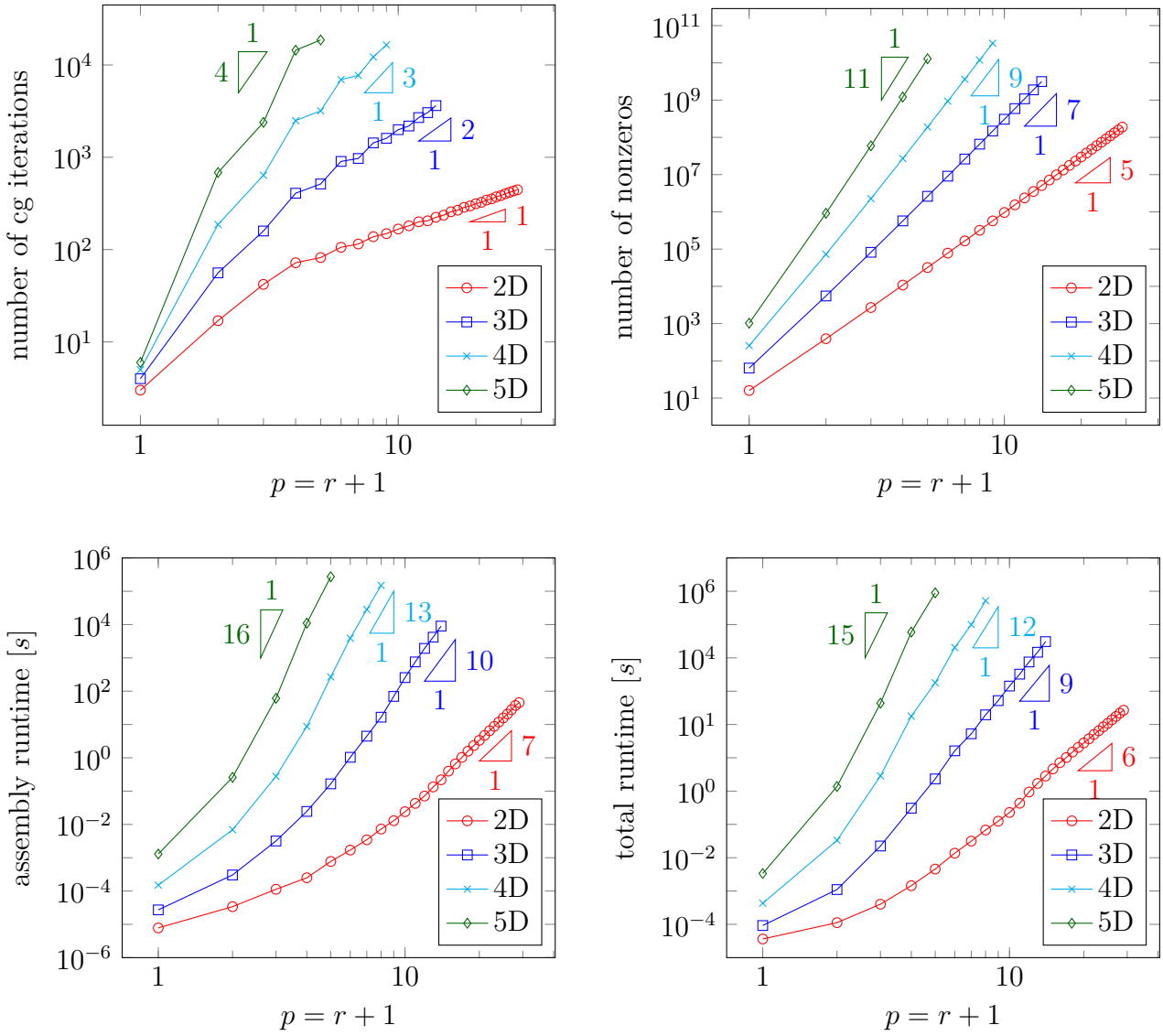


Figure 5.4: Number of CG iterations, number of non-zeros in the sparse matrix, and serial runtime comparison for the uniformly graded tensor spaces. The total runtime excludes the error computation. From Kopp, Rank, et al. (2022), used under Creative Commons CC BY 4.0 license.

analysis shows that the method achieves exponential convergence in terms of these crucial quantities. The finite element linear systems are solved using a diagonally preconditioned conjugate gradient (CG) method. Therefore, the number of non-zeros in the matrix dominates the memory consumption. The two most significant contributions to the runtime are the assembly and the solution of the linear system.

The asymptotic behavior of these quantities is analyzed in terms of the refinement study index r to allow comparisons with runtime measurements. For brevity, only the full tensor-product space using a uniform polynomial degree elevation is considered. Each new refinement level adds 2^d cells that overlay the cell in the origin of the parent level. Therefore, the number of elements (leaf cells) can be computed as $r(2^d - 1) + 1 = \mathcal{O}\big(r\big)$.

Each leaf cell contributes to at most $(p+1)^d \sim p^d$ basis functions defined on the same refinement level, but it also receives contributions from all coarser levels. The key to obtaining good scaling is that parent cells only contribute basis functions active on their faces. This results in $(p+1)^d - (p-1)^d \sim p^{d-1}$ functions per (coarser) level at most. Because $r \sim p$, these contributions scale with

$$\mathcal{O}\left(rp^{d-1}\right) \underset{r\sim p}{\longrightarrow} \mathcal{O}\left(p^d\right)$$

and hence are of the same order as the $(p-1)^d = \mathcal{O}\left(p^d\right)$ internal functions coming from the classical p-version of the finite element method.

The number of non-zeros in the assembled matrix can be bounded to some value between the sum of all element matrix sizes (upper bound) and the sum of all internal contributions (lower bound). Using $\mathcal{O}\left(rp^{d-1}+p^d\right)$ as an upper bound for the number of element unknowns and multiplying with $O(r)$ number of elements yields

$$\mathcal{O}\left(r\right)\mathcal{O}\left(rp^{d-1}+p^d\right)^2 \underset{r\sim p}{\longrightarrow} \mathcal{O}\left(p^{2d+1}\right)$$

as an estimate of the complexity for the number of non-zeros in the assembled sparse matrix. The same result is obtained when considering only internal contributions.

Similarly, the assembly effort is estimated by summing up the number of operations needed to integrate each element matrix numerically. Assuming a standard Gauss-Legendre quadrature, one needs $(p+1)^d = \mathcal{O}\left(p^d\right)$ integration points, on which an outer product of $\mathcal{O}\left(rp^{d-1}+p^d\right)$ shape functions is evaluated, resulting in the following estimate:

$$\mathcal{O}\left(r\right)\mathcal{O}\left(p^d\right)\mathcal{O}\left(rp^{d-1}+p^d\right)^2 \underset{r\sim p}{\longrightarrow} \mathcal{O}\left(p^{3d+1}\right).$$

Figure 5.4 verifies that the number of non-zeros in the sparse matrix scales as derived above, while the number of CG iterations increases roughly with p^{d-1}; the analysis of this empirical observation depends on the condition number of the sparse matrix that exceeds the scope of this work. The iterative solver terminates when $\sqrt{\boldsymbol{R}\cdot D(\boldsymbol{R})} < 10^{-10}$, where $D(\boldsymbol{R}) = \mathrm{diag}(\mathbf{K})^{-1}\boldsymbol{R}$ is the diagonal preconditioner applied to the residual $\boldsymbol{R} = \mathbf{K}\hat{\boldsymbol{u}} - \boldsymbol{F}$. Moreover, the symmetry of the sparse matrix is not utilized, which significantly simplifies the parallel implementation of the sparse matrix-vector product. By combining the results for the number of non-zeros with the number of iterations, the runtime complexity for the iterative solver is estimated as $\mathcal{O}(p^{3d})$. Figure 5.4 shows that the performance measurements verify these theoretical estimates. Although the assembly of the linear system scales with the highest order, it only starts significantly contributing to the total runtime towards the end of the curves. Moreover, direct solvers with a linear complexity with respect to the number of unknowns exist for such types of meshes (Paszynski et al., 2012).

The measurements change significantly when considering the trunk space with the graded polynomial degree distribution. While the overall scaling is the same, the absolute effort per degree of freedom is higher, and the relative runtime of the assembly increases compared to the linear solution. There are two main reasons for these differences. First, although the fine levels were assigned a lower polynomial degree, they still receive contributions from the higher-order base elements. Due to these, even the finest level with $p = 1$ requires $r + 2$

Gauss-Legendre points per direction to integrate the parent contributions accurately. Second, removing internal modes increases the connectivity in the trunk space, resulting in higher runtime and increased memory consumption per unknown. When considering two interacting shape functions, N_i and N_j, the effort for assembling this interaction is proportional to the number of elements N_i and N_j overlap. Internal functions overlap only on one element but functions active on the faces overlap at least two elements, making internal functions cheaper to assemble. After the assembly, however, this interaction results in a single entry in the sparse matrix regardless of the number of elements the supports overlap.

5.2 AMB2018-02 benchmark[2]

Laser parameters	
Speed (v)	$0.8\,\mathrm{m\,s^{-1}}$
Power (P)	179.2 W
Absorptivity (ν)	0.32
Spot size ($D4\sigma$)	170 µm
Depth (σ_z)	$0.28 \cdot D4\sigma/4$

Phase change parameters	
Latent heat (L)	$2.8 \times 10^5\,\mathrm{J\,kg^{-1}}$
Solid temperature (u_s)	1290 °C
Liquid temperature (u_l)	1350 °C
Initial temperature (u_0)	25 °C
Regularization (S)	1, 2, 4

Table 5.1: Model parameters (see Sections 2.1 and 2.2). From Kopp, Calo, et al. (2022), used under Creative Commons CC BY 4.0 license.

	Fine discretization				Coarse discretization			
Base mesh	$64 \times 64 \times 9 \times 32$ slabs				$12 \times 12 \times 3 \times 8$ slabs			
Refinement	τ [ms]	d_τ	σ_τ [µm]	z-factor	τ [ms]	d_τ	σ_τ [µm]	z-factor
	0	6	100	0.5	0	5.4	180	0.5
	0.082	5.3	120	0.5	1.2	3.5	240	0.5
	0.47	4.5	150	0.5	6	2.5	400	0.8
	1.2	3.5	160	0.8	30	1.5	900	1
	4	2.5	200	1	100	1	1100	1
	16	0.5	240	1				
Polynomial degrees in space	2 (uniform)				2, 2, 4, 4, 3 and 3 (for levels 0 to 5, respectively)			

Table 5.2: Fine and coarse discretization parameters. From Kopp, Calo, et al. (2022), used under Creative Commons CC BY 4.0 license.

The AMB2018-02 benchmark (National Institute of Standards and Technology, 2018) is commonly used to initially assess the performance of new simulation methods for PBF-LB/M processes. The setup features various laser configurations in single strokes on a block of IN625 alloy ($24.08\,\mathrm{mm} \times 24.82\,\mathrm{mm} \times 3.18\,\mathrm{mm}$). The length of each stroke is 14 mm with a duration of 17.5 ms, but the melt pool usually reaches a steady state after at most 2 mm of travel distance.

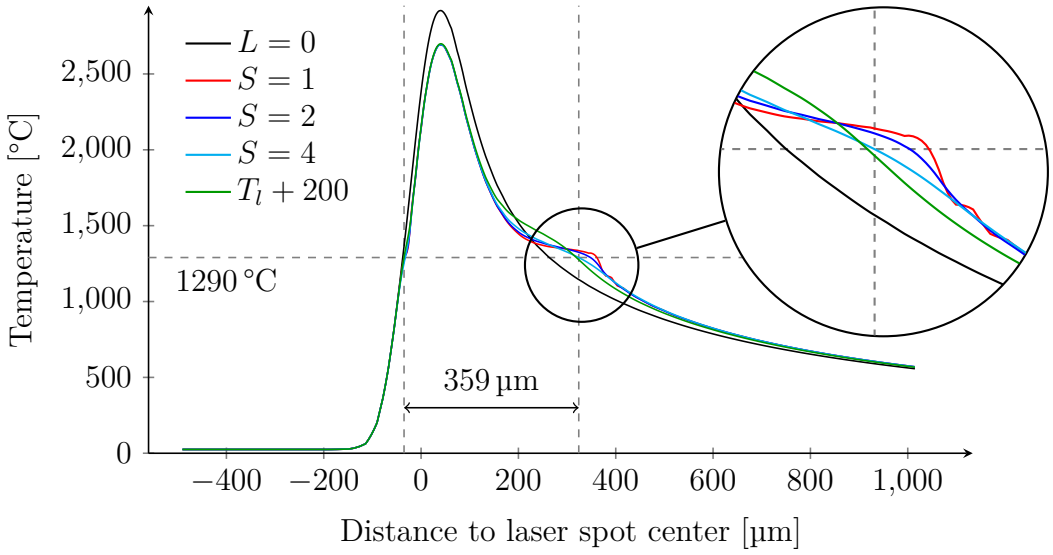


Figure 5.5: Temperature in laser travel direction for different phase change parameters. From Kopp, Calo, et al. (2022), used under Creative Commons CC BY 4.0 license.

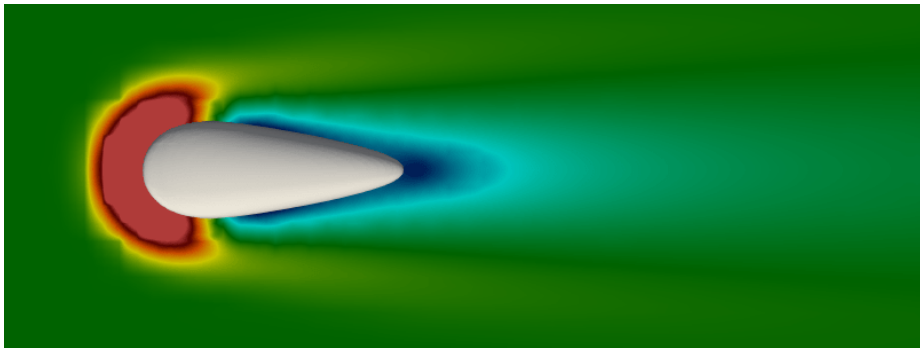
Experimental data from Lane et al. (2020) shows the melt pool width and depth for each track as measured from the cross-section area within which the microstructure changed. The melt pool length and cooling rates are estimated using high refresh rate thermal imaging. This section focuses on test case B on the additive manufacturing metrology testbed (AMMT) and the measurements from Lane et al. (2020, Table 3) for track number 3. Table 5.1 summarizes the simulation model parameters. The thermal properties of IN625 are

$$c_s(u) = \left(405 + \frac{247 \cdot u}{1000\,°\mathrm{C}}\right)\left[\frac{\mathrm{J}}{\mathrm{kg}\,°\mathrm{C}}\right] \qquad u \le u_s$$

$$k(u) = \left(9.5 + \frac{15 \cdot u}{1000\,°\mathrm{C}}\right)\left[\frac{\mathrm{W}}{\mathrm{m}\,°\mathrm{C}}\right] \qquad u \le u_s$$

for temperatures below the melting point and are assumed constant for higher temperatures. All simulations use a constant mass density $\rho = 8440\,\mathrm{kg\,m^{-3}}$.

In the first numerical experiments, the model is calibrated to reproduce the experimental data as closely as possible. The melt pool dimensions are estimated from the simulation results by extracting a contour surface at $u = u_s$ and measuring its bounding box's length, width, and depth. The first calibration step identifies suitable values for the absorptivity ν and the penetration depth σ_z of the heat source in (2.2) and (2.3) such that the width and depth match those of the experiment. Then, the phase change regularization is adjusted for the melt pool length to match the experiments. This strategy uses the property that increasing the smoothness of the phase change mainly affects the melt pool length while having a minor effect on its width and depth. By using a well-resolved discretization, as seen in Table 5.2, the discretization influence is minimized. All computations use linear polynomials in time combined with a trunk space for the spatial discretization. The additional z-factor in Table 5.2 multiplies the refinement width σ_τ in the z-direction to prevent unnecessary refinement in the region below the melt pool.



(a) Top view

(b) Side view

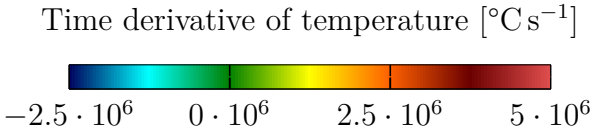


Figure 5.6: Time derivative of the temperature field using the fine discretization. From Kopp, Calo, et al. (2022), used under Creative Commons CC BY 4.0 license.

Result	Length [µm]	Width [µm]	Depth [µm]
Measurements	359 ($\sigma = 20$)	132 ($\sigma = 2$)	36 ($\sigma = 0.9$)
No latent heat	301	138	39.4
$S = 1$	396	129	34.8
$S = 2$	381	129	34.8
$S = 4$	356	129	34.8
$S = 8$	328	130	34.8
$u_l + 200$	354	131	35.4
$u_l + 200$ (coarse)	353	132	35.3

Table 5.3: Melt pool dimensions for different model parameters in comparison to experimental data. From Kopp, Calo, et al. (2022), used under Creative Commons CC BY 4.0 license.

Table 5.3 shows the results for different phase change regularizations, where $\nu = 0.32$ and $\sigma_z = 0.28 \cdot D4\sigma/4$. The dimensions for $S = 4$ (equivalent to $u_s = 1170\,°\text{C}$ and $u_l = 1470\,°\text{C}$) and with only increasing u_l to 1550 °C are very similar. However, Figure 5.5 shows that the regularization of the phase transition below 1290 °C significantly changes the cooling rate after the solidification. This influence may interfere with a prediction of the microstructure forma-

tion, for which a regularization towards higher temperatures may be preferable. Figure 5.6 shows the time derivative of the temperature field obtained directly from the space-time finite element discretization. The cooling rates of around $2 \times 10^6\,°\mathrm{C\,s^{-1}}$ deviate significantly from the measured $1.08 \times 10^6\,°\mathrm{C\,s^{-1}}$, given by Lane et al. (2020, Table 3). However, in Lane et al. (2020), the authors advise against using the cooling rate measurements due to motion blur and a limited calibration range. Therefore, the validation of cooling rates for this thermal model remains an open task. Next, the discretization is coarsened as much as possible while

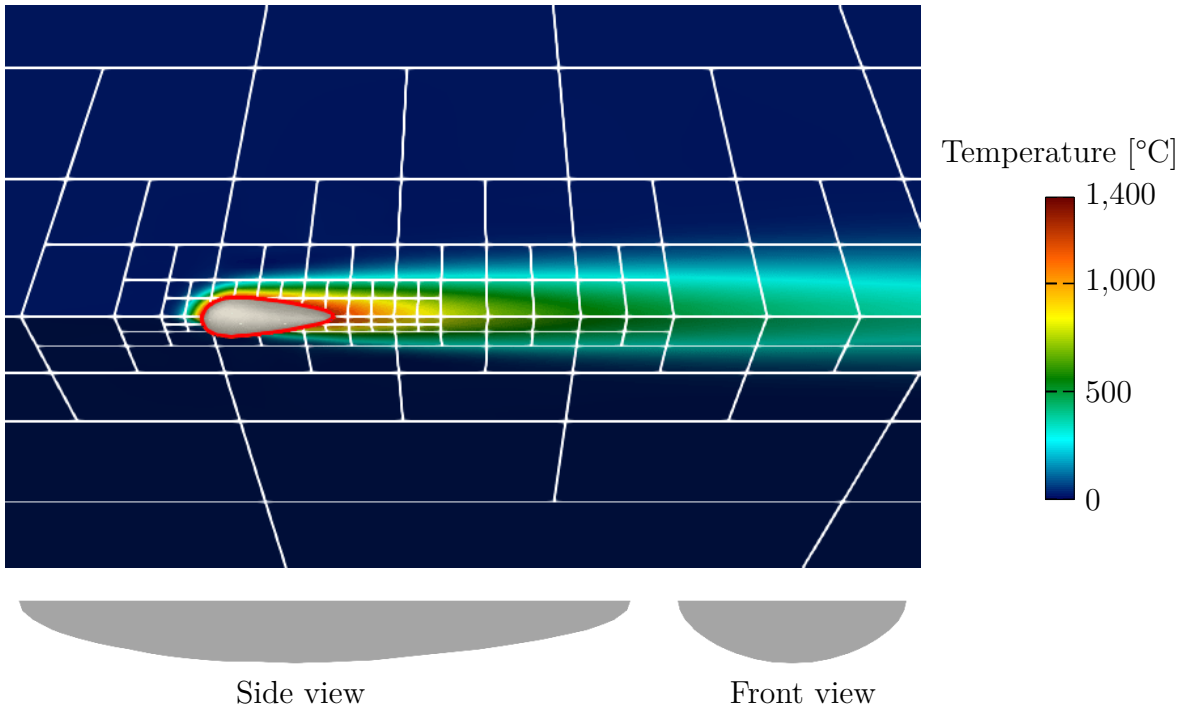


Figure 5.7: Solution and contour surface at $u_s = 1290\,°\mathrm{C}$ for coarse discretization with $u_l = 1550\,°\mathrm{C}$. From Kopp, Calo, et al. (2022), used under Creative Commons CC BY 4.0 license.

still obtaining good estimates to showcase the benefits of the presented approach. Figure 5.7 shows the melt pool geometry and the temperature in the vicinity for the coarse discretization; see Table 5.2. A simulation time of 17.5 ms and eight time slabs results in a duration of about 2.2 ms for one slab. The melt pool dimensions (last row of Table 5.3) are almost identical to the ones of the well-resolved discretization. For simplicity, all elements are over-integrated with $p+2$ quadrature points in space and $p+1$ quadrature points in time to capture the phase change accurately. The linear systems are solved with Intel's Pardiso sparse direct solver.

5.3 Hatched square[2]

This section uses the setup of the previous section for the AMB2018-02 benchmark to hatch a square area with a 10 mm side length. As Figure 5.8 shows, the laser path first follows the contour line and then fills the interior with a hatch distance of 100 µm, resulting in a path length of about 102 cm. The entire process takes about 1.28 s, which is extended to a total simulation time of 3 s. During this cooldown period, the discretization is automatically coarsened in space and time as a result of the formulation and the way the meshes are constructed,

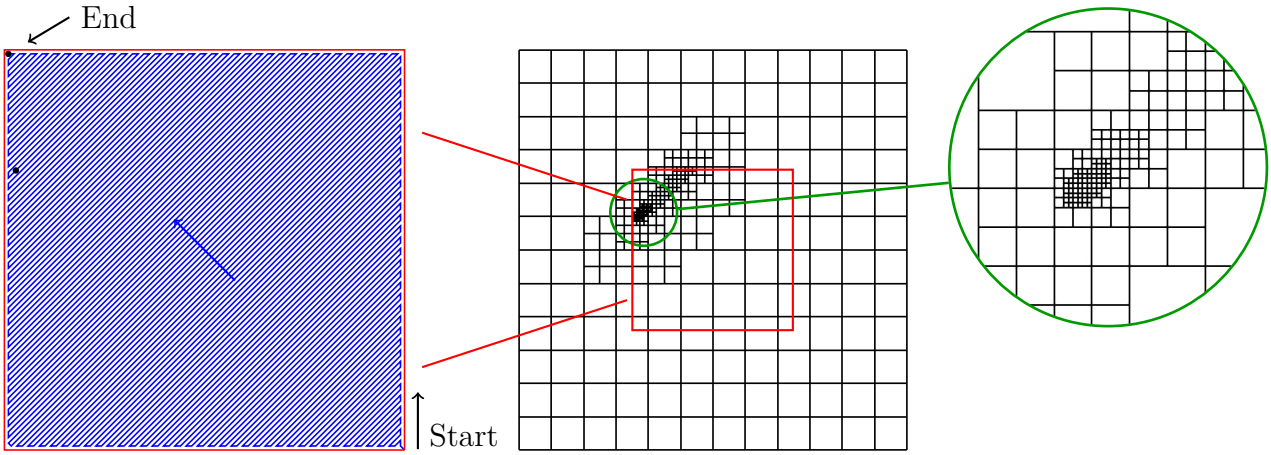


Figure 5.8: Laser path (left) and example discretization (center and right) at $t = 1.2168\,\mathrm{s}$ for a hatched square with 10 mm side length and 100 µm hatch width. From Kopp, Calo, et al. (2022), used under Creative Commons CC BY 4.0 license.

as Section 4.5 discusses. Each time slab contains one base element in time with a duration of 2.4 ms, which is slightly longer than in the previous example. The number of unknowns per slab initially averages around 50 thousand and drops to around 2400 for slabs in the cooldown period.

Figure 5.9 shows the solution and the melt pool dimensions for two time slices; on the left side, the laser approaches the top left corner, and on the right side, it has just reached it. Significant heating of the plate appears, leading to more than a 50% predicted increase in melt pool width and depth for the second case. Hence, this spot is identified as a potential source of defects.

The simulation runs on a single Intel Xeon Gold 6230 CPU with 20 cores running at 2.1 GHz in about 7 hours. Compared to the single-threaded execution, the parallel version is about eight times faster, a good result considering that the CPU's turbo boost frequency is 3.9 GHz, which, from experience, results in a maximum speedup of 11 to 12 times. While this speedup is reached in the assembly of the linear systems, the Pardiso sparse direct solver does not scale optimally in the examples. Moreover, the assembly again over-integrates with $p + 2$ points in space and $p + 1$ points in time. This performance penalty can be improved by identifying the elements around the laser source and increasing the number of quadrature points there only.

5.4 Performance comparison to time-stepping

This section compares the locally refined space-time discretization in four dimensions to a more conventional three-dimensional locally refined *hp*-approach. The computational domain in space is initially chosen as $[-8\,\mathrm{mm}, 8\,\mathrm{mm}]^3$ and is discretized with a base mesh of two elements per direction. The initial domain and discretization are then gradually extended to $[-43.2\,\mathrm{cm}, 43.2\,\mathrm{cm}]^3$ and a base mesh of 108 elements per direction. Both discretization types are refined seven levels towards the laser spot using the same level function to enable a meaningful comparison. The time interval is in all cases $[0\,\mathrm{ms}, 8.75\,\mathrm{ms}]$. Figure 5.10 shows the increasingly large spatial meshes at $t = 8.75\,\mathrm{ms}$. The time-stepping reference uses 128

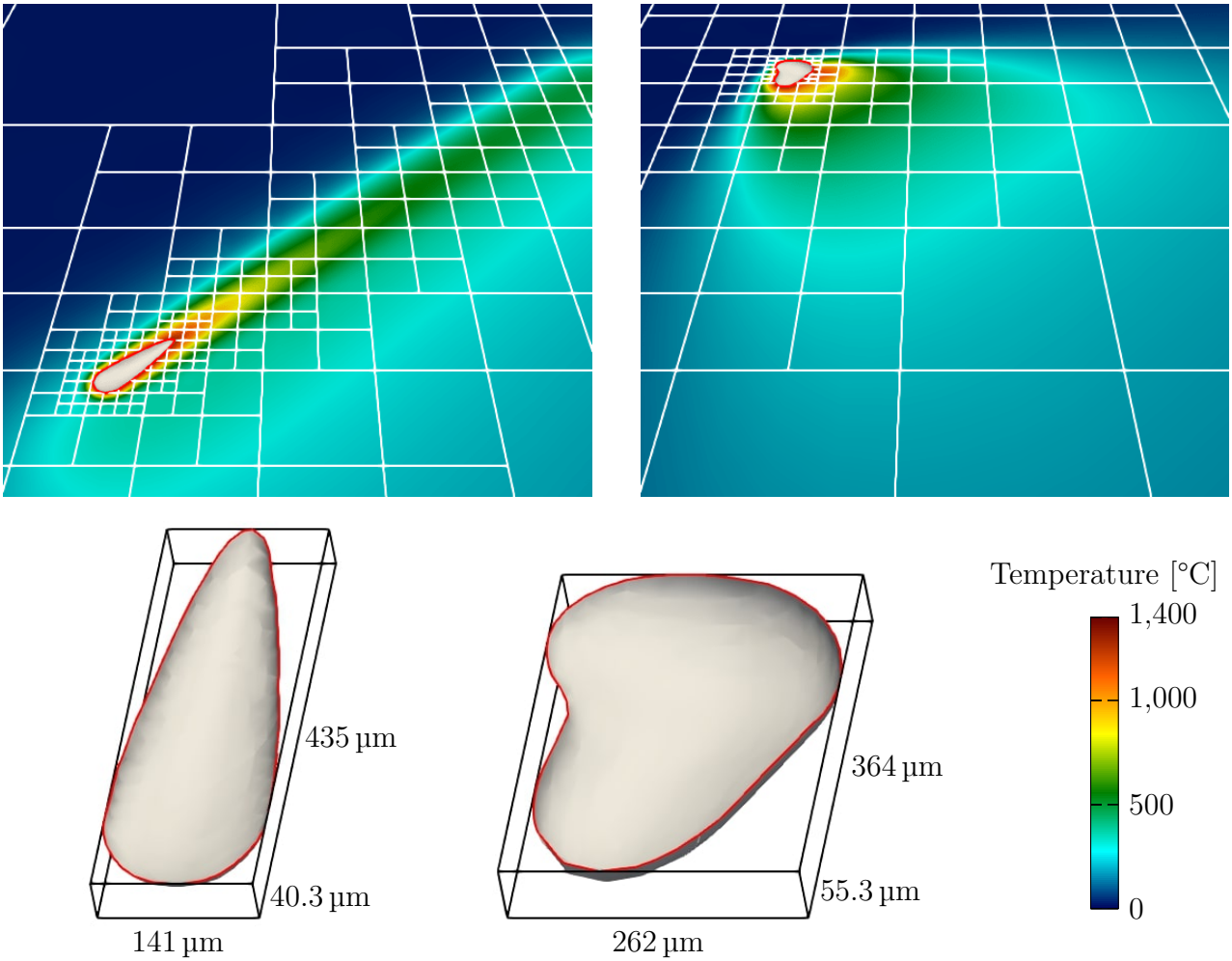
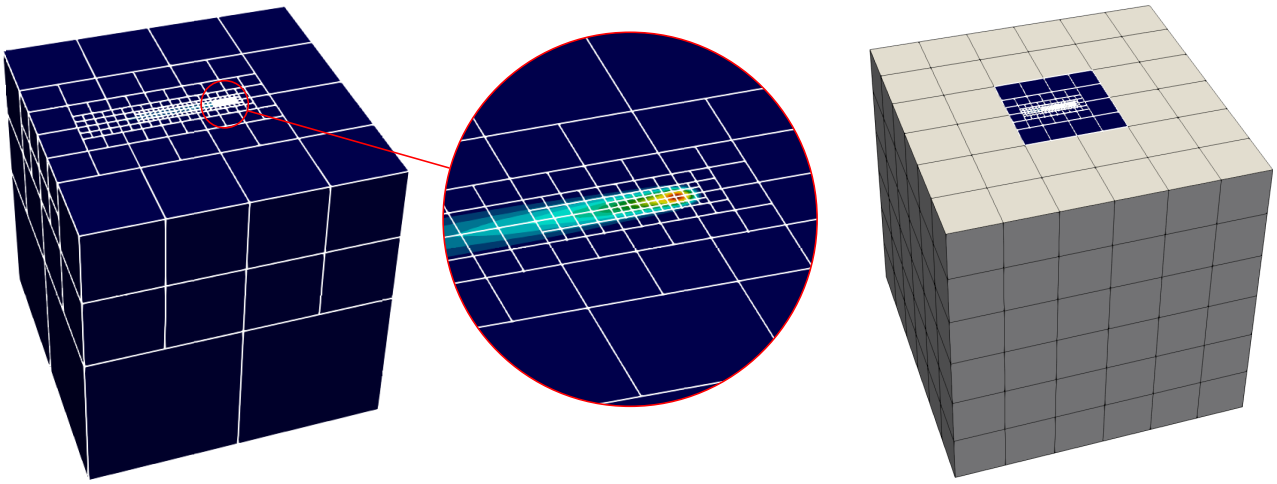


Figure 5.9: Temperature and melt pool geometry for $t = 1.2168\,\mathrm{s}$ (left) and $t = 1.2786\,\mathrm{s}$ (right). From Kopp, Calo, et al. (2022), used under Creative Commons CC BY 4.0 license.

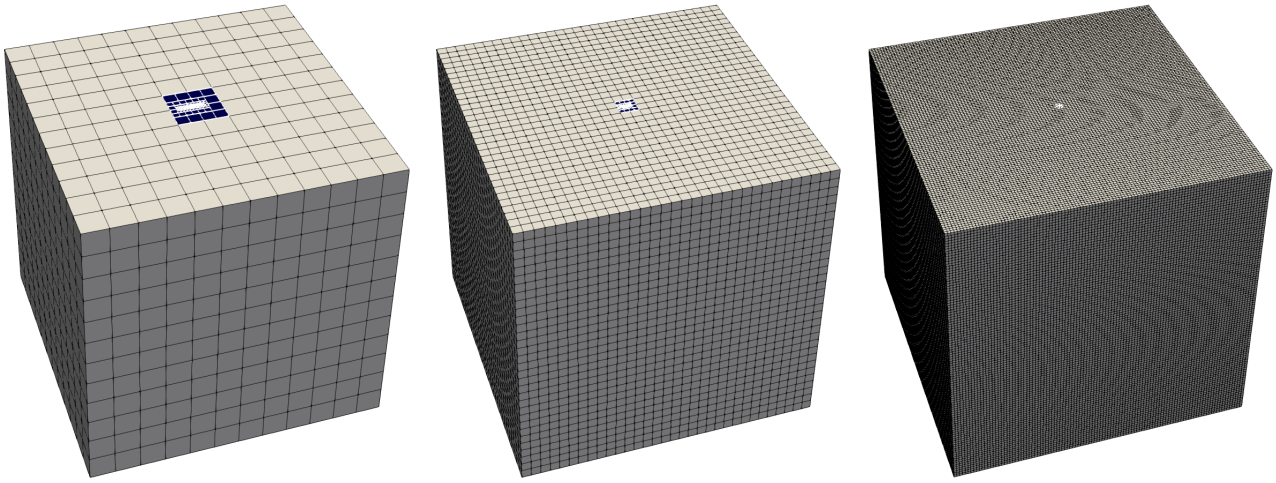
Crank-Nicolson steps ($\theta = 1/2$), while the space-time discretization reduces the element durations automatically by refining locally in four dimensions. On the finest elements, the time interval is identical; however, the space-time mesh also coarsens in time, away from the laser. Both discretizations use the trunk space with polynomial degree 3 in space. The space-time discretization uses a linear interpolation in time.

In Figure 5.11, the number of unknowns for the space-time mesh is initially much larger than for a single time step mesh. There are only slightly fewer degrees of freedom in the time slab mesh than in the meshes of all time steps together. However, as the domain is extended and more elements are added, the peripheral part of the domain starts to dominate the number of unknowns until, eventually, a single time step is as expensive as a single time slab.

The relative reduction of the residual norm during the nonlinear iterations of each time step reaches the threshold value of 10^{-10} in about ten iterations (further iterations stagnate at 10^{-13} to 10^{-11}). The nonlinear iterations of the single time slab systems of the space-time discretization converge to a relative residual norm reduction threshold of 10^{-14} in about seven iterations (stagnating at around 10^{-16}). The linear systems of the space-time discretization are solved using the Intel Pardiso sparse direct solver. From now on, this approach is called ST-Pardiso.



(a) Smallest domain with 1.6 cm width (b) Extension to 4.8 cm width



(c) Extension to 9.6 cm width (d) Extension to 25.6 cm width (e) Extension to 86.4 cm width

Figure 5.10: Benchmark with a small laser stroke in the center and an increasing peripheral cost resulting from adding root elements on the left, right, front, back, and bottom sides. Each mesh is refined seven times. The solution shown is evaluated at $t = 8.75$ ms.

The linear systems of the time-stepping discretizations are solved using two methods: first, also using Intel's Pardiso solver (TS-Pardiso) to compare to ST-Pardiso directly, and second, using a diagonally preconditioned CG method combined with the sparse matrix-vector multiplication of the Intel MKL library (TS-CG). All three approaches (ST-Pardiso, TS-Pardiso, TS-CG) are computed in serial and also using shared-memory (Open-MP) parallelism with 20 cores/threads (again on a single Intel Xeon Gold 6230 CPU). The TS-CG approach uses the standard (unsymmetric) compressed sparse row (CSR) format to store sparse matrices. Although the symmetric version of the CSR format that stores only half of the matrix performs better in the serial case, it is not further investigated in this section to simplify the comparisons. Similarly, TS-Pardiso uses the same setup as ST-Pardiso, where an unsymmetric CSR matrix is given to the Pardiso solver without "mentioning" the symmetry of the time-stepping formulation. Potentially faster versions based on either symmetric CSR matrices or unsym-

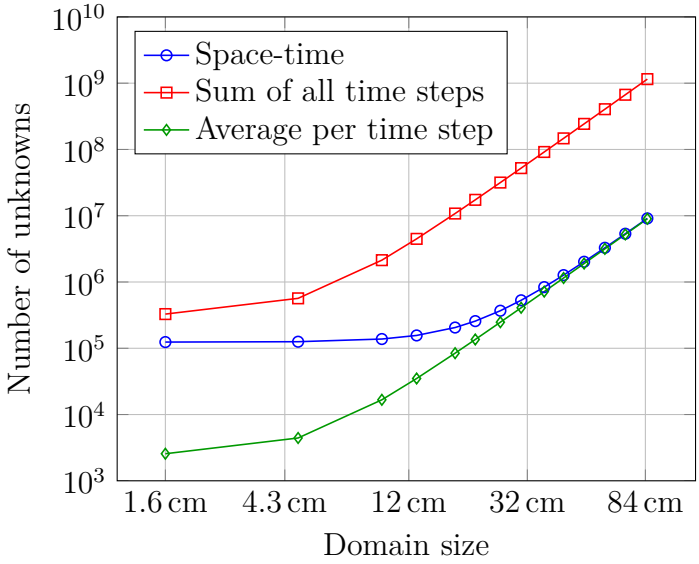


Figure 5.11: Comparison between single space-time slab and 128 time steps.

metric CSR matrices with a consistent linearization (2.12) of (2.11) (leading to fewer nonlinear iterations) are excluded from the following comparison. The inconsistent linearization used by TS-Pardiso and TS-CG does not affect the initial iterations, but it slows the convergence afterwards and requires more iterations to reach machine precision. Hence, choosing a higher threshold value can reduce the difference in the number of nonlinear iterations between ST-Pardiso and TS-Pardiso/CG. Moreover, the space-time finite element integrals are integrated with four quadrature points per spatial direction ($p + 1$) and one quadrature point in time (p). In contrast, the previous sections used one additional quadrature point per direction to improve the accuracy of the results. While these choices may slightly skew the results in favor of the space-time discretization, they do not change the general characteristics discussed in the following.

Figure 5.12 compares the serial and parallel runtime for ST-Pardiso, TS-Pardiso, and TS-CG approaches. The serial runtime of the space-time discretization is initially about twice as long as TS-Pardiso and TS-CG. Although the number of unknowns of the single slab is lower than the sum of all time steps, the solution of a single large equation system is more expensive than solving many smaller systems. Moreover, the overlap of basis functions in four dimensions is higher than in three, further increasing the runtime and memory consumption. When expanding the domain, the *peripheral computational cost* dominates much later for the space-time discretization, leading to a similar runtime (compared to the initial mesh) for much longer than the time-stepping. While this holds for TS-Pardiso and TS-CG, the iterative solver in TS-CG scales better than the direct solver for ST-Pardiso and TS-Pardiso. When extending the domain, the number of CG iterations does not increase much (i.e., less than five percent). While this example uses constant initial data, the same holds for spatially varying initial temperature fields, where the time step lengths are still so small that the peripheral temperature increments are almost zero. As a result, the runtime of TS-CG eventually catches up to ST-Pardiso once the linear systems become very large. The same results can be observed in the OMP-parallel execution but with two differences. First, on the large initial mesh, all three approaches require roughly the same amount of time. Figure 5.13 shows that the parallel

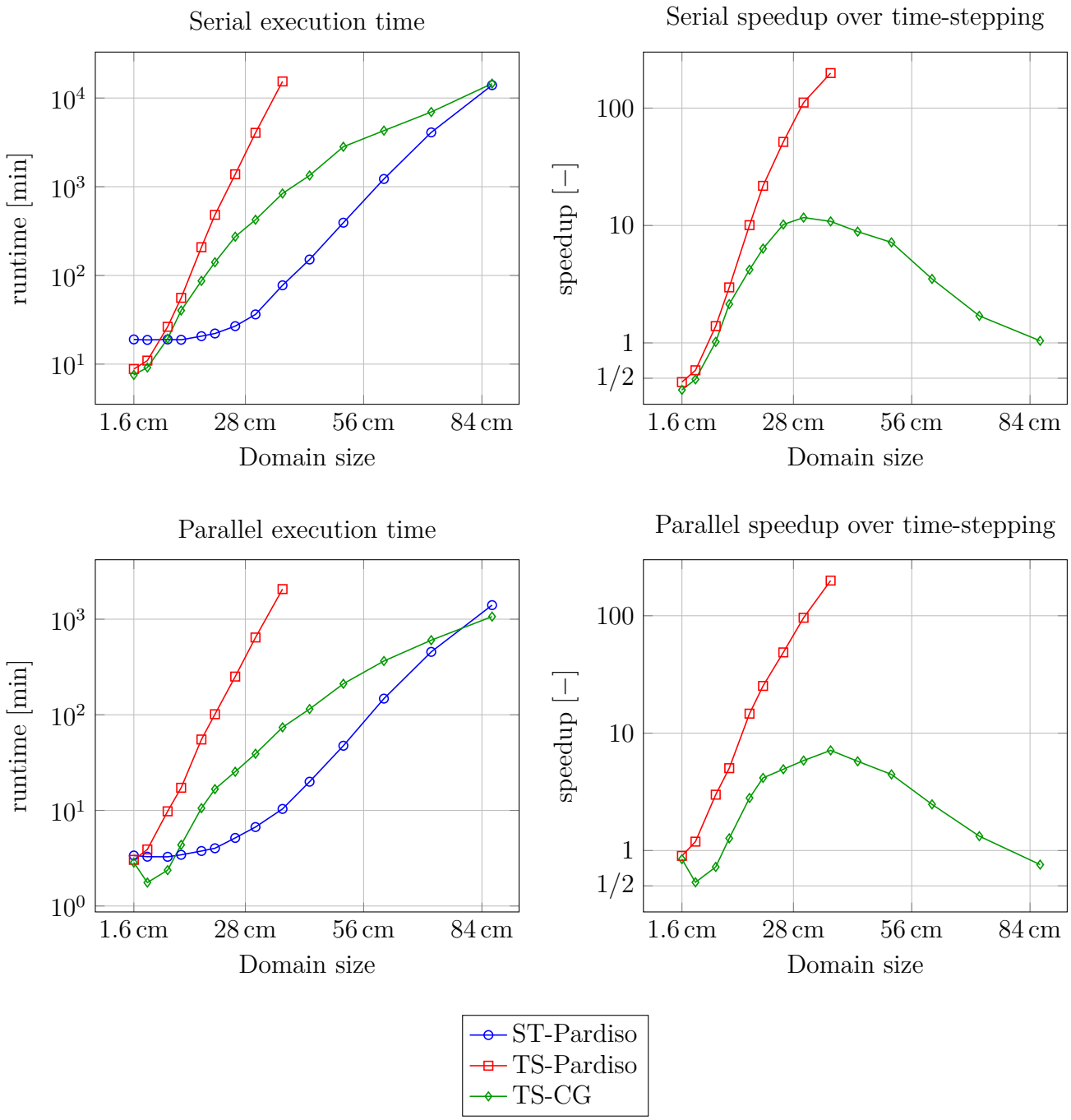


Figure 5.12: Comparison between single space-time slab and 128 time steps.

efficiency of ST-Pardiso is initially much better than TS-CG and, in particular, TS-Pardiso. The second difference to the serial execution is the improved parallel scaling of the TS-CG method for bigger meshes, as the Intel MKL's sparse matrix-vector multiplication scales better than their sparse direct solver. An interesting characteristic of TS-CG is that later Newton-Raphson iterations with an almost converged solution require fewer CG iterations in the solution of the linear system, as Figure 5.14 shows.

These results show that the space-time discretization has the potential to capture the multi-

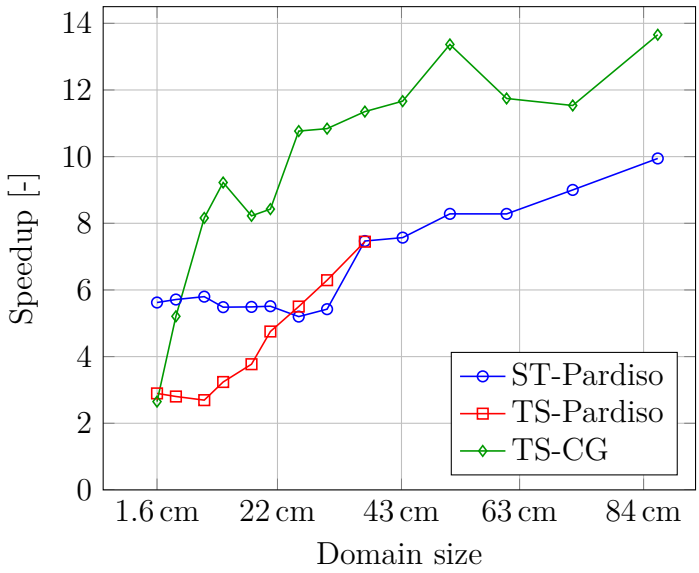


Figure 5.13: Speedup of parallel vs. serial execution.

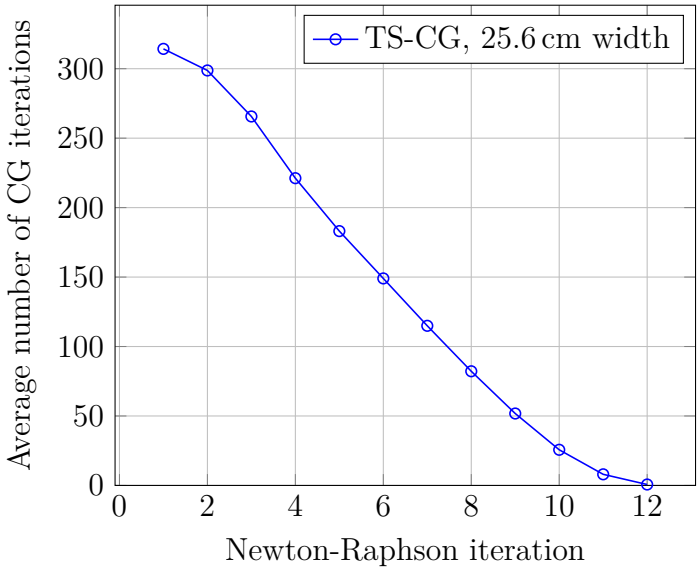


Figure 5.14: Average number of CG iterations per Newton-Raphson iteration for TS-CG.

scale nature of the thermal evolution of PBF-LB/M processes better than conventional time-stepping methods. The advantage of local refinement in four dimensions grows with an increased peripheral computational cost. While in this example, more elements were added on the boundary of the base mesh, more realistic simulations can cause an increased peripheral cost in other ways. For instance, resolving complex geometries based on a very heterogeneous material history using immersed methods can be computationally expensive.

Chapter 6

Conclusion

This thesis presents a space-time multi-level *hp*-method to approximate the nonlinear heat equation for simulating the temperature evolution in PBF-LB/M processes. Local refinements in four dimensions are possible by using a multi-level *hp*-framework to construct data-oriented algorithms and data structures that extend to any number of dimensions. The introduction of tensor-product masks and location matrices with corresponding pair-wise operations on their array slices provides the foundation for efficient implementations. In particular, a small memory management overhead and favorable memory access patterns distinguish this approach from the original object-oriented formulation already in one, two, and three dimensions. While the simplicity of the presented alternative reduces the effort to construct and maintain the *hp*-finite element data structures, modifying the mesh topology invalidates basis data that was computed on the previous mesh state. However, applications with evolving meshes often benefit from creating an entirely new mesh and basis rather than modifying the existing ones, which results in cleaner implementations and can simplify applications that require both the new and the old discretization, such as certain time-stepping schemes or error estimators.

The combination of this multi-level *hp* extension with a continuous Galerkin-Petrov formulation of the nonlinear heat equation leads to a finite element method that discretizes space and time together in a four-dimensional setting. Testing with the time derivative of the trial functions results in optimal convergence and allows splitting the solution into consecutive time slabs, which is crucial for maintaining manageable problem sizes. By meshing one time slab ahead, the multi-level *hp*-basis functions are compatible across the slab interface in time by construction. The refinement of the time slab meshes starts at the current position of the laser with the maximum refinement depth. It gradually reduces towards positions further in the past while simultaneously increasing the refinement width. The thermal model uses a volumetric source that extends the two-dimensional laser intensity profile into the material by a Gaussian function in the z-direction. The length of this extension allows tuning the width and depth of the resulting melt pool shape to the experiments. Moreover, the latent heat of fusion is modeled as a spike in the heat capacity. Different regularization shapes of the phase transition primarily influence the cooling rates and the length of the simulated melt pool.

In the first preliminary verification benchmark, the convergence of the multi-level *hp* extension is analyzed using a manufactured solution to Poisson's equation with a point singularity. When increasing the refinement depth and the polynomial degree, the energy error reduces exponentially while maintaining a polynomial runtime and memory scaling. These results agree with the theoretical estimates and are essential for obtaining good performance in real-world applications. Then, the space-time formulation is validated against the experimental

results published for the AMB2018-02 benchmark. The dimensions of the simulated melt pool shape coincide with the measurements after tuning the absorptivity, the depth extension of the laser source, and the shape of the phase change regularization. Shifting the latent heat capacity spike towards higher temperatures reduces the influence on the temperatures after the solidification. A significant difference in the cooling rates after solidification may result from measurement inaccuracies and requires further investigation. The potential for computing longer and more complicated paths is demonstrated using the setup of the AMB2018-02 benchmark to hatch an area of one square centimeter with diagonal strokes. The ad hoc refinement captures the solution characteristic well, and the resulting temperature field provides insight into problematic regions with larger than intended melt pools. Finally, the runtime of the space-time approach is compared to an equivalent second-order time-stepping method. This study starts with a small domain that is gradually extended by adding base elements on the boundary. This increase in peripheral cost impacts the space-time approach less because the base mesh of one time slab uses only a single element in time for a relatively long duration. The speedup over the equivalent time-stepping can reach factors of ten or even over one hundred when using a direct solver for both approaches. A major current limitation of the presented space-time method is the unsymmetric linear system and the need to use a direct solver.

This thesis shows that finite element discretizations with local refinement in space-time can bridge the scales present in the simulation of PBF-LB/M processes. Such proof of concept is a crucial first step, but several challenges must be addressed before the presented approach can be used in practice. Longer simulations require including boundary conditions that account for the heat loss due to radiation and convection. The simulation of an actual powder bed must also consider the reduced conductivity of metal powder compared to melted and solidified metal. This history dependency may pose additional challenges in combination with a space-time discretization. Then, the speedup compared to time-stepping methods may be significant in some situations, but it is insufficient to enable part-scale simulations. Massively parallel implementations are necessary to efficiently use enough HPC resources and reduce the computational time to a practical amount. The extensions of the multi-level *hp* data structures and the development of a scalable iterative solver are crucial for the success of the method. Alternative to investigating linear solvers for the presented Petrov-Galerkin method, exploring different space-time methods that are more suitable for parallel extensions may be necessary. A potential application for such an efficient and scalable thermal solver for PBF-LB/M processes may be optimizing the laser power along a given laser path. Automatically creating a time-discrete laser power that minimizes constraints like a constant melt pool volume with minimal deviations from the CAD model may help improve and automate the 3D printing workflow. Finally, automatic adaptive methods can capture complicated solution characteristics of real applications more efficiently and robustly than the presented ad hoc refinement based on the laser path history.

Bibliography

Akrivis, G., Makridakis, C., & Nochetto, R. (2011). Galerkin and Runge–Kutta methods: Unified formulation, a posteriori error estimates and nodal superconvergence. *Numerische Mathematik*, *118*, 429–456. https://doi.org/10.1007/s00211-011-0363-6

Aziz, A., & Monk, P. (1989). Continuous finite elements in space and time for the heat equation. *Mathematics of Computation*, *52*, 255–274. https://doi.org/10.1090/S0025-5718-1989-0983310-2

Babuska, I., Szabo, B. A., & Katz, I. N. (1981). The p-version of the finite element method. *SIAM Journal on Numerical Analysis*, *18*(3), 515–545. https://doi.org/10.1137/0718033

Bayat, M., Dong, W., Thorborg, J., To, A., & Hattel, J. (2021). A review of multi-scale and multi-physics simulations of metal additive manufacturing processes with focus on modeling strategies. *Additive Manufacturing*, *47*, 102278. https://doi.org/10.1016/j.addma.2021.102278

Bayat, M., Thanki, A., Mohanty, S., Witvrouw, A., Yang, S., Thorborg, J., Tiedje, N. S., & Hattel, J. H. (2019). Keyhole-induced porosities in laser-based powder bed fusion (L-PBF) of Ti6Al4V: High-fidelity modelling and experimental validation. *Additive Manufacturing*, *30*, 100835. Licensed under CC BY-NC-ND 4.0. https://doi.org/10.1016/j.addma.2019.100835

Blakey-Milner, B., Gradl, P., Snedden, G., Brooks, M., Pitot, J., Lopez, E., Leary, M., Berto, F., & Du Plessis, A. (2021). Metal additive manufacturing in aerospace: A review. *Materials & Design*, *209*, 110008. https://doi.org/10.1016/j.matdes.2021.110008

Buchanan, C., & Gardner, L. (2019). Metal 3D printing in construction: A review of methods, research, applications, opportunities and challenges. *Engineering Structures*, *180*, 332–348. https://doi.org/10.1016/j.engstruct.2018.11.045

Celentano, D., Oñate, E., & Oller, S. (1994). A temperature-based formulation for finite element analysis of generalized phase-change problems. *International Journal for Numerical Methods in Engineering*, *37*(20), 3441–3465. https://doi.org/10.1002/nme.1620372004

Cheng, L., & Wagner, G. J. (2021). An optimally-coupled multi-time stepping method for transient heat conduction simulation for additive manufacturing. *Computer Methods in Applied Mechanics and Engineering*, *381*, 113825. https://doi.org/10.1016/j.cma.2021.113825

Demkowicz, L. (2006). *Computing with hp-adaptive finite elements, vol. 1: One and two dimensional elliptic and Maxwell problems*. Chapman & Hall/CRC Applied Mathematics & Nonlinear Science.

Demkowicz, L., Oden, J., Rachowicz, W., & Hardy, O. (1989). Toward a universal *h-p* adaptive finite element strategy, part 1. constrained approximation and data structure. *Computer*

Methods in Applied Mechanics and Engineering, *77*(1), 79–112. https://doi.org/10.1016/0045-7825(89)90129-1

Devaud, D., & Schwab, C. (2018). Space–time *hp*-approximation of parabolic equations. *Calcolo*, *55*. https://doi.org/10.1007/s10092-018-0275-2

Di Stolfo, P., Schröder, A., Zander, N., & Kollmannsberger, S. (2016). An easy treatment of hanging nodes in *hp*-finite elements. *Finite Elements in Analysis and Design*, *121*, 101–117. https://doi.org/10.1016/j.finel.2016.07.001

Egorov, S., Khmyrov, R., Korotkov, A., & Gusarov, A. (2020). Experimental study and modeling of melt pool in laser powder-bed fusion of thin walls. *Procedia CIRP*, *94*, 372–377. https://doi.org/10.1016/j.procir.2020.09.148

Elhaddad, M., Zander, N., Bog, T., Kudela, L., Kollmannsberger, S., Kirschke, J., Baum, T., Ruess, M., & Rank, E. (2017). Multi-level *hp*-finite cell method for embedded interface problems with application in biomechanics. *International Journal for Numerical Methods in Biomedical Engineering*, *34*. https://doi.org/10.1002/cnm.2951

Eriksson, K., Johnson, C., & Thomée, V. (1985). Time discretization of parabolic problems by the discontinuous Galerkin method. *Journal of Multivariate Analysis - MA*, *19*. https://doi.org/10.1051/m2an/1985190406111

Ern, A., & Guermond, J.-L. (2004). *Theory and practice of finite elements*. New York, NY, Springer New York. https://doi.org/10.1007/978-1-4757-4355-5

Führer, T., & Karkulik, M. (2021). Space–time least-squares finite elements for parabolic equations. *Computers & Mathematics with Applications*, *92*, 27–36. https://doi.org/10.1016/j.camwa.2021.03.004

Gordon, J. V., Narra, S. P., Cunningham, R. W., Liu, H., Chen, H., Suter, R. M., Beuth, J. L., & Rollett, A. D. (2020). Defect structure process maps for laser powder bed fusion additive manufacturing. *Additive Manufacturing*, *36*, 101552. Licensed under CC BY-NC-ND 4.0. https://doi.org/10.1016/j.addma.2020.101552

Grünewald, J., Gehringer, F., Schmoeller, M., & Wudy, K. (2021). Influence of ring-shaped beam profiles on process stability and productivity in laser-based powder bed fusion of AISI 316L. *Metals*, *11*, 1989. https://doi.org/10.3390/met11121989

Gu, D., Shi, Q., Lin, K., & Xi, L. (2018). Microstructure and performance evolution and underlying thermal mechanisms of Ni-based parts fabricated by selective laser melting. *Additive Manufacturing*, *22*. https://doi.org/10.1016/j.addma.2018.05.019

Gusarov, A., Yadroitsev, I., Bertrand, P., & Smurov, I. (2009). Model of radiation and heat transfer in laser-powder interaction zone at selective laser melting. *Journal of Heat Transfer*, *131*, 072101. https://doi.org/10.1115/1.3109245

Harkin, R., Wu, H., Nikam, S., Yin, S., Lupoi, R., McKay, W., Walls, P., Quinn, J., & McFadden, S. (2022). Powder reuse in laser-based powder bed fusion of Ti6Al4V—changes in mechanical properties during a powder top-up regime. *Materials*, *15*(6). Licensed under CC BY 4.0. https://doi.org/10.3390/ma15062238

Hodge, N. (2021). Towards improved speed and accuracy of laser powder bed fusion simulations via representation of multiple time scales. *Additive Manufacturing*, *37*, 101600. https://doi.org/10.1016/j.addma.2020.101600

Hofer, C., Langer, U., Neumüller, M., & Toulopoulos, I. (2018). Time-multipatch discontinuous Galerkin space-time isogeometric analysis of parabolic evolution problems. *ETNA - Electronic Transactions on Numerical Analysis*, *49*, 126–150. https://doi.org/10.1553/etna_vol49s126

Hughes, T. (2000). *The finite element method: Linear static and dynamic finite element analysis*. Dover Publications.

Hussain, S., Schieweck, F., & Turek, S. (2011). Higher order Galerkin time discretizations and fast multigrid solvers for the heat equation. *Journal of Numerical Mathematics*, *19*. https://doi.org/10.1515/JNUM.2011.003

Imani Shahabad, S., Zhang, Z., Keshavarzkermani, A., Ali, U., Mahmoodkhani, Y., Esmaeilizadeh, R., Bonakdar, A., & Toyserkani, E. (2020). Heat source model calibration for thermal analysis of laser powder-bed fusion. *The International Journal of Advanced Manufacturing Technology*, *106*, 1–13. https://doi.org/10.1007/s00170-019-04908-3

ISO 17296-2:2015(E). (2015). *Additive manufacturing – general principles – overview of process categories and feedstock* (Standard). International Organization for Standardization. Geneva, CH.

Jamet, P. (1978). Galerkin-type approximations which are discontinuous in time for parabolic equations in a variable domain. *SIAM Journal on Numerical Analysis*, *15*, 912–928. https://doi.org/10.1137/0715059

Jomo, J., de Prenter, F., Elhaddad, M., D'Angella, D., Verhoosel, C., Kollmannsberger, S., Kirschke, J., Nübel, V., Van Brummelen, H., & Rank, E. (2018). Robust and parallel scalable iterative solutions for large-scale finite cell analyses. *Finite Elements in Analysis and Design*, *163*. https://doi.org/10.1016/j.finel.2019.01.009

Jomo, J., Oztoprak, O., de Prenter, F., Zander, N., Kollmannsberger, S., & Rank, E. (2021). Hierarchical multigrid approaches for the finite cell method on uniform and multi-level *hp*-refined grids. *Computer Methods in Applied Mechanics and Engineering*, *386*, 114075. https://doi.org/10.1016/j.cma.2021.114075

Khairallah, S. A., Anderson, A. T., Rubenchik, A., & King, W. E. (2016). Laser powder-bed fusion additive manufacturing: Physics of complex melt flow and formation mechanisms of pores, spatter, and denudation zones. *Acta Materialia*, *108*, 36–45. https://doi.org/10.1016/j.actamat.2016.02.014

Kollmannsberger, S., Carraturo, M., Reali, A., & Auricchio, F. (2019). Accurate prediction of melt pool shapes in laser powder bed fusion by the non-linear temperature equation including phase changes—isotropic versus anisotropic conductivity. *Integrating Materials and Manufacturing Innovation*, *8*, 167–177. https://doi.org/10.1007/s40192-019-00132-9

Kollmannsberger, S., & Kopp, P. (2021). On accurate time integration for temperature evolutions in additive manufacturing. *GAMM-Mitteilungen*, *44*. https://doi.org/10.1002/gamm.202100019

Kollmannsberger, S., Özcan, A., Carraturo, M., Zander, N., & Rank, E. (2017). A hierarchical computational model for moving thermal loads and phase changes with applications to selective laser melting. *Computers & Mathematics with Applications*, *75*. https://doi.org/10.1016/j.camwa.2017.11.014

Kopp, P., Calo, V., Rank, E., & Kollmannsberger, S. (2022). Space-time *hp*-finite elements for heat evolution in laser powder bed fusion additive manufacturing. *Engineering with Computers*. Licensed under CC BY 4.0. https://doi.org/10.1007/s00366-022-01719-1

Kopp, P., Rank, E., Calo, V., & Kollmannsberger, S. (2022). Efficient multi-level *hp*-finite elements in arbitrary dimensions. *Computer Methods in Applied Mechanics and Engineering*, *401*, 115575. Licensed under CC BY 4.0. https://doi.org/10.1016/j.cma.2022.115575

Lane, B., Heigel, J., Ricker, R., Zhirnov, I., Khromschenko, V., Weaver, J., Phan, T., Stoudt, M., Mekhontsev, S., & Levine, L. (2020). Measurements of melt pool geometry and cooling rates of individual laser traces on IN625 bare plates. *Integrating Materials and Manufacturing Innovation*, *9*. https://doi.org/10.1007/s40192-020-00169-1

Langer, U., Matculevich, S., & Repin, S. (2019). 5. Adaptive space-time isogeometric analysis for parabolic evolution problems. De Gruyter. https://doi.org/10.1515/9783110548488-005

Langer, U., & Yang, H. (2020). BDDC preconditioners for a space-time finite element discretization of parabolic problems. International Conference on Domain Decomposition Methods. https://doi.org/10.1007/978-3-030-56750-7_42

Letenneur, M., Brailovski, V., Kreitcberg, A., Paserin, V., & Bailon-Poujol, I. (2017). Laser powder bed fusion of water-atomized iron-based powders: Process optimization. *Journal of Manufacturing and Materials Processing*, *1*(2). Licensed under CC BY 4.0. https://doi.org/10.3390/jmmp1020023

Loli, G., Sangalli, G., & Tesini, P. (2022). High-order spline upwind for space-time isogeometric analysis. arXiv. https://doi.org/10.48550/ARXIV.2211.02692

Meidner, D., & Vexler, B. (2007). Adaptive space-time finite element methods for parabolic optimization problems. *SIAM Journal on Control and Optimization*, *46*(1), 116–142. https://doi.org/10.1137/060648994

National Institute of Standards and Technology. (2018). *AMB2018-02 description*. Retrieved November 4, 2021, from https://www.nist.gov/ambench/amb2018-02-description

Nitzler, J., Meier, C., Müller, K., Wall, W., & Hodge, N. (2021). A novel physics-based and data-supported microstructure model for part-scale simulation of laser powder bed fusion of Ti-6Al-4V. *Advanced Modeling and Simulation in Engineering Sciences*, *8*. https://doi.org/10.1186/s40323-021-00201-9

Paszynski, M., Calo, V., & Pardo, D. (2012). A direct solver with reutilization of LU factorizations for h-adaptive finite element grids with point singularities. *Computers & Mathematics with Applications*, *65*. https://doi.org/10.1016/j.camwa.2013.02.006

Paulson, N., Gould, B., Wolff, S., Stan, M., & Greco, A. (2020). Correlations between thermal history and keyhole porosity in laser powder bed fusion. *Additive Manufacturing*, *34*, 101213. https://doi.org/10.1016/j.addma.2020.101213

Ranjan, R., Ayas, C., Langelaar, M., & van Keulen, F. (2020). Fast detection of heat accumulation in powder bed fusion using computationally efficient thermal models. *Materials*, *13*(20). https://doi.org/10.3390/ma13204576

Rausch, A. M., Küng, V. E., Pobel, C., Markl, M., & Körner, C. (2017). Predictive simulation of process windows for powder bed fusion additive manufacturing: Influence of the powder bulk density. *Materials*, *10*(10). Licensed under CC BY 4.0. https://doi.org/10.3390/ma10101117

Rongzeng, Y., Luo, D., Huang, H., Li, R., Yu, N., Liu, C., Hu, M., & Rong, Q. (2018). Electron beam melting in the fabrication of three-dimensional mesh titanium mandibular prosthesis scaffold. *Scientific Reports*, *8*. Licensed under CC BY 4.0. https://doi.org/10.1038/s41598-017-15564-6

Schieweck, F. (2010). A-stable discontinuous Galerkin–Petrov time discretization of higher order. *Journal of Numerical Mathematics*, *18*, 25–57. https://doi.org/10.1515/JNUM.2010.002

Schmich, M., & Vexler, B. (2008). Adaptivity with dynamic meshes for space-time finite element discretizations of parabolic equations. *SIAM Journal on Scientific Computing*, *30*(1), 369–393. https://doi.org/10.1137/060670468

Schwab, C., & Stevenson, R. (2009). Space–time adaptive wavelet methods for parabolic evolution problems. *Math. Comput.*, *78*, 1293–1318. https://doi.org/10.1090/S0025-5718-08-02205-9

Soldner, D., & Mergheim, J. (2019). Thermal modelling of selective beam melting processes using heterogeneous time step sizes. *Computers & Mathematics with Applications*, *78*(7), 2183–2196. https://doi.org/10.1016/j.camwa.2018.04.036

Steinbach, O. (2015). Space-time finite element methods for parabolic problems. *Computational Methods in Applied Mathematics*, *15*. https://doi.org/10.1515/cmam-2015-0026

Steinbach, O., & Yang, H. (2019). 7. Space-time finite element methods for parabolic evolution equations: Discretization, a posteriori error estimation, adaptivity and solution. In U. Langer & O. Steinbach (Eds.), *Applications to partial differential equations* (pp. 207–248). Berlin, Boston, De Gruyter. https://doi.org/doi:10.1515/9783110548488-007

Szabo, B. A., & Mehta, A. K. (1978). P-convergent finite element approximations in fracture mechanics. *International Journal for Numerical Methods in Engineering*, *12*(3), 551–560. https://doi.org/10.1002/nme.1620120313

Trapp, J., Rubenchik, A., Guss, G., & Matthews, M. (2017). In situ absorptivity measurements of metallic powders during laser powder-bed fusion additive manufacturing. *Applied Materials Today*, *9*, 341–349. https://doi.org/10.1016/j.apmt.2017.08.006

Viguerie, A., Carraturo, M., Reali, A., & Auricchio, F. (2022). A spatiotemporal two-level method for high-fidelity thermal analysis of laser powder bed fusion. *Finite Elements in Analysis and Design*, *210*, 103815. https://doi.org/10.1016/j.finel.2022.103815

Xie, X., Bennett, J., Saha, S., Lu, Y., Cao, J., Liu, W., & Gan, Z. (2021). Mechanistic data-driven prediction of as-built mechanical properties in metal additive manufacturing. *npj Computational Materials*, *7*, 86. https://doi.org/10.1038/s41524-021-00555-z

Yadroitsev, I., Yadroitsava, I., Du Plessis, A., & Macdonald, E. (2021). *Fundamentals of laser powder bed fusion of metals*. Elsevier. https://doi.org/10.1016/B978-0-12-824090-8.00004-4

Zander, N., Bériot, H., Hoff, C., Kodl, P., & Demkowicz, L. (2022). Anisotropic multi-level *hp*-refinement for quadrilateral and triangular meshes. *Finite Elements in Analysis and Design*, *203*, 103700. https://doi.org/10.1016/j.finel.2021.103700

Zander, N., Bog, T., Elhaddad, M., Frischmann, F., Kollmannsberger, S., & Rank, E. (2016). The multi-level *hp*-method for three-dimensional problems: Dynamically changing high-order mesh refinement with arbitrary hanging nodes. *Computer Methods in Applied Mechanics and Engineering*, *310*, 252–277. https://doi.org/10.1016/j.cma.2016.07.007

Zander, N., Bog, T., Kollmannsberger, S., Schillinger, D., & Rank, E. (2015). Multi-level *hp*-adaptivity: High-order mesh adaptivity without the difficulties of constraining hanging nodes. *Computational Mechanics*, *55*(3), 499–517. https://doi.org/10.1007/s00466-014-1118-x

Zhang, Z., Huang, Y., Rani Kasinathan, A., Imani Shahabad, S., Ali, U., Mahmoodkhani, Y., & Toyserkani, E. (2018). 3-Dimensional heat transfer modeling for laser powder-bed fusion additive manufacturing with volumetric heat sources based on varied thermal conductivity and absorptivity. *Optics & Laser Technology*, *109*, 297–312. https://doi.org/10.1016/j.optlastec.2018.08.012